SIMATIC S5
CP 521 BASIC
Communications Processor

Manual

EWA 4NEB 812 6071-02a

Edition 02

STEP ® und SIMATIC ® are registered trademarks of Siemens AG.

Copyright © Siemens AG 1991

Subject to change without prior notice.

The reproduction, transmission or use of this document or its contents is not permitted without express written authority. Offenders will be liable for damages. All rights, including rights created by patent grant or registration of a utility model or design, are reserved.

Preface	
Introduction	
System Overview	**1**
Technical Description	**2**
Installation Guidelines	**3**
Principles of Operation and Addressing	**4**
BASIC Interpreter	**5**
Communications Interfaces	**6**
CPU Requests and Error Recovery Procedures	**7**
Programs	**8**
Appendix	**A**
Appendix	**B**
Keyword Index	

Preface

The CP 521 BASIC communications processor is a powerful I/O module and an integral component of the S5-100U programmable controller. Detailed information is needed to put this module to optimum use.

Because the performance capabilities of the S5-100U are being continually upgraded, the S5-100U manual is already extremely extensive.
For this reason, a separate manual was prepared for the CP 521 BASIC module. This manual was designed to provide only information and examples pertinent to the CP 521 BASIC module yet still satisfy the increasing demands placed on technical documentation. These include:

- The standardization of terminology and notation
- Extensive section and subsection breakdowns
- Visual guidance
- Customer-tailored presentation

The primary objective of the manual is to provide both beginners and experienced SIMATIC S5 users will all the information they need to use the CP 521 BASIC communications processor.

It is not possible, however, to cover every conceivable problem in a single manual. Should you wish further information, please contact your local Siemens branch office for help.

Introduction

Before continuing, please read the introduction through carefully. This will help you use the manual, and will also save you time.

Modifications and improvements to the CP 521 BASIC, 6ES5 521-8MB11, version 6 and the CP 521 BASIC, 6ES5 521-8MB12, version 1 compared to previous versions of the CP 521 BASIC.

- Automatic restart of the CP 521 BASIC using the new AUTOINIT function
  Until now, if the CP 521 BASIC's battery backup failed (or if a battery was not in use), it was not possible to start up the CP 521 BASIC without connecting a terminal (entering SPACE). With Version 6 of the CP 521 BASIC, it is possible to perform a restart without battery backup.

- Extending the ONERR function
  The ONERR statement enables a programming solution to arithmetic errors in the BASIC program. If an arithmetic error occurs, such as division by zero, the program is continued in accordance with the line number specified in the ONERR statement.

  Until now, it was only possible to evaluate the cause of error (error code). There was, however, no possibility of finding the number of the line in which the error occurred. The extension of the ONERR function now enables this line number to be found using the XBY statement. The address of the line number in which the error occurred is at:

  600 (high byte)
  601 (low byte)

Description of contents

The manual provides a detailed description of the CP 521 BASIC module. It is subdivided into sections, each section dealing with a specific topic.

- Description
  - The ”System Overview” deals with the tasks which the module can perform and how the module is used within the S5-100U system.
  - The ”Technical Description” provides general information on how the module functions, and also includes all technical specifications, information on the serial interfaces and I/Os, and a list of available accessories.

- Installation and operation
  - The ”Installation Guidelines” provide all information needed to install the module and interface it to a printer or other I/O device (connector pin assignments and connection diagrams).

- Principles of operation and addressing
  - This section provides general information on how data interchange between CPU, CP 521 BASIC and I/Os is coordinated and on
  - the module's restart performance and how the module is addressed.

- Functional description
  - The section headed ”BASIC Interpreter” contains all information needed to write a program, including descriptions of all BASIC instructions.
  - The section entitled ”Communications Interfaces” provides detailed information on data interchange between the CPU, the CP 521 BASIC module and an I/O device. This section is aimed primarily at users who want to program their own data interchange FBs, but also provides information on troubleshooting.

- Error recovery
  The section entitled ”CPU Requests and Error Recovery” covers all CPU requests which can be issued in BASIC mode. It lists all errors reported by the BASIC interpreter and provides information on error recovery procedures.

- Programs
  The section headed ”BASIC Interpreter” contains all information needed to write a program, including descriptions of all BASIC instructions.

- Appendix A:
  The appendix provides lists in tabular form of all BASIC instructions and auxiliary instructions for communication with the CPU and the I/Os to simplify use of the CP 521 BASIC.

Conventions

For the purpose of clarity, the manual is organized in menu form, i.e.:

- Each section has a thumb index.
- The manual begins with an overview of the section headers.
- Each section is preceded by a section breakdown.
  There is a three-level breakdown for each section. Additional breakdowns are preceded by headers in bold type.
- Pages, figures and tables are numbered separately in each section. The figures and tables in a section are listed on the flip side of the page containing the breakdown for that section.

Certain conventions were observed when writing the manual. These are explained below.

- A number of abbreviations have been used.
  Example: Programmer (PG)
- Footnotes are identified by superscripts consisting of a small digit (e.g. ”[1]”), or ”*”. The actual footnote is generally at the bottom left of the page or below the relevant table or figure.
- Cross references are shown as follows:
  ”(7.2.1)” refers to section 7.2.1.
  No references are made to individual pages
- All dimensions in drawing etc. are given in millimetres followed by inches in brackets.
- Information of particular importance is framed in grey-bordered rectangles.
- All programming examples have been generated in statement list form and always refer to the same slot.
- All programmer specifications refer to the German versions of the PG 635, PG 685, PG 730 and PG 750. See the relevant Operator's Guide for a detailed description of programming with these programmers.

Manuals can only describe the current version of the device. Should modifications or supplements become necessary in the course of time, a supplement will be prepared and included in the manual the next time it is revised. The relevant version or edition of the manual appears on the cover: in this case, edition ”1”. In the event of a revision, the edition number will be incremented by ”1”.

Please use the forms at the back of the manual for any suggestions or corrections you may have and return the forms to us. This will help us to make the necessary improvements in the next edition.

Safety-Related Guidelines for the User

This document provides the information required for the intended use of the CP 521 BASIC. The documentation is written for technically qualified personnel.

Qualified personnel as understood by the safety-related guidelines printed in this document or on the product itself are defined as being

- either system planning and design engineers who are familiar with the safety concept of automation equipment;
- or operating personnel who have been trained to work with automation equipment and are conversant with the contents of this document;
- or system start-up and service personnel who have been trained to repair such automation equipment and are authorized to start up, ground and tag circuits, equipment and systems in keeping with the relevant safety standards and codes of practice.

Danger notices

The following notices are intended, on the one hand, for your personal safety and, on the other, to prevent damage to the product described or equipment connected to it.

The safety notices and warnings highlighted in this manual by the terms and pictograms defined here are intended for the personal safety of both the user and his or her service personnel, as well as to prevent damage to property and equipment. The terms used in this manual and marked on the product itself have the following significance:

Warning

indicates that death, severe personal injury or substantial damage to property can result if the proper precautions are not taken.

Note

indicates important information about the product or the respective part of the instruction manual to which special attention is drawn.

Bestimmungsgemäßer Gebrauch

Warning

- This equipment may only be used for the applications listed in the catalog and technical description and only in conjunction with products and components of other vendors recommended by Siemens.
- Satisfactory and safe operation of the equipment can only be guaranteed if it is properly handled (transported), stored and installed, and operated and maintained with the appropriate care.

Courses

Siemens provide SIMATIC S5 users with extensive opportunities for training. For more information, please contact your Siemens representative.

Training courses

SIEMENS offers a wide variety of training courses for SIMATIC S5 users.

For details, please contact your local Siemens branch office.

Reference literature

This manual provides a detailed description of the CP 521 BASIC communications processor. Topics which do not pertain specifically to the CP 521 BASIC module are discussed only superficially. More detailed information can be found in the following publications:

- **Programmable Controls**
  Volume 1: Logic and sequence controls; from the control problem to control program

  Günter Wellenreuther, Dieter Zastrow
  Brunswick 1987

  Contents:
  - Theory of operation of a programmatic control system
  - Theory of logic control technology using the STEP 5 programming language for SIMATIC S5 programmable controllers.

  Order No.: ISBN 3-528-04464-0

- **Automating with the S5-115U**
  SIMATIC S5 Programmable Controllers

  Hans Berger
  Second revised edition, Berlin; Munich: Siemens AG, 1989

  Contents:
  - STEP 5 programming language
  - Program scanning
  - Integral blocks
  - Interfaces to the I/Os

  Order No.: ISBN 3-89578-022-7

- **Programming Primer for the SIMATIC S5-100U**
  Practical Exercises with the PG 615 Programmer

  Siemens AG, Berlin and Munich, 1988

  Contents:
  - Configuring and Installing the S5-100U Programmable Controller
  - Introduction to Programming with the PG 615 Programmer

  Order No.: ISBN 3-8009-1528-6

1 System Overview

1 System Overview

The CP 521 BASIC is a powerful active I/O module which can be installed in all S5-100U programmable controllers (except those with CPU 100s, 6ES5 100-8MA01 up to revision level 5). The module, which is equipped with its own CPU, is available together with a special COM software package, which is required for writing and archiving (FLOPPY, EPROM) BASIC programs.

A BASIC interpreter allows the user to write and execute BASIC programs which can interchange data with the CPU and an interfaced I/O device. The BASIC interpreter is programmed on a programmer (PG) or terminal using the COM software. The programs can be stored in the module's battery-backed RAM or on a plug-in memory submodule.

The programmer or terminal is connected to the CP 521 BASIC over the latter's serial TTY (20 mA current-loop) or RS232C (V.24) interface. A printer can be connected to the module's unidirectional RS232C (V.24) interface for printing out listings or logging messages.
All signals for the TTY and RS232C (V.24) interface are applied to a common connector.

The parameters for the I/O interface can be modified either via BASIC instructions or in the BASIC program.

2 Technical Description

2.1 Approbations and Tests . . . . . 2 - 1

2.2 Notes for the Machine Manufacturer . . . . . 2 - 3

2.3 Technical Specifications. . . . . 2 - 4

2.4 Memory Submodules . . . . . 2 - 5

2.5 Serial Interfaces . . . . . 2. - 6

2.6 Integral Real-Time Clock . . . . . 2 - 7

2.7 Backup Battery . . . . . 2. - 7

2.8 LEDs . . . . . 2. - 8

2.9 List of Accessories and Order Numbers . . . . . 2 - 8

Tables

2-1. Erasing and Storing Data on Memory Submodules . . . . . 2 - 5
2-2. Overview of Permissible Memory Submodules . . . . . 2 - 5
2-3. Overview of Permissible Memory Submodules that Can Still be Used . . . . . 2. - 6

2 Technical Description

2.1 Approbations and Tests

The general technical specifications include standards and test specifications which the CP 521 BASIC meets and fulfills and which were used during testing of the CP 521 BASIC.

UL/CSA Approbations

The following approbations have been granted for the CP 521 BASIC:
UL-Recognition Mark
Underwriters Laboratories (UL) to UL standard 508, Report 116536
CSA Certification Mark
Canadian standard Association (CSA) to C22.2 standard No. 142, Report LR 48323

CE-Marking

Our products meet the requirements of EU directive 89/336/EEC "Electromagnetic Compatibility" and the harmonized European standards (EN) listed therein.

CE In accordance with the above-mentioned EU directive, Article 10, the EU declarations of conformity are held at the disposal of the competent authorities at the address below:

Siemens Aktiengesellschaft
Bereich Automatisierungstechnik
AUT E 14
Postfach 1963

D-92209 Amberg
Federal Republic of Germany

Area of Application

SIMATIC products have been designed for use in the industrial area.
With individual approval, SIMATIC products can also be used in the domestic environment (household, business and trade area, small plants). You must acquire the individual approval from the respective national authority or testing board.

Area of Application	Requirements to:	
	Emitted interference	Immunity
Industry	EN 50081-2 : 1993	EN 50082-2 : 1995
Domestic	Individual approval	EN 50082-1 : 1992

Observing the Installation Guidelines

S5 modules meet the requirements if you observe the installation guidelines described in the manuals when installing and operating the equipment (Section 3).

2.2 Notes for the Machine Manufacturer

The SIMATIC programmable controller system is not a machine as defined in the EU Machinery Directive. There is therefore no declaration of conformity for SIMATIC with regard to the EU Machinery Directive 89/392/EEC.

The EU Machinery Directive 89/392/EEC regulates requirements relating to machinery. A machine is defined here as an assembly of linked parts or components (see also EN 292-1, Paragraph 3.1).

SIMATC is part of the electrical equipment of a machine and must therefore be included by the machine manufacturer in the declaration of conformity procedure.

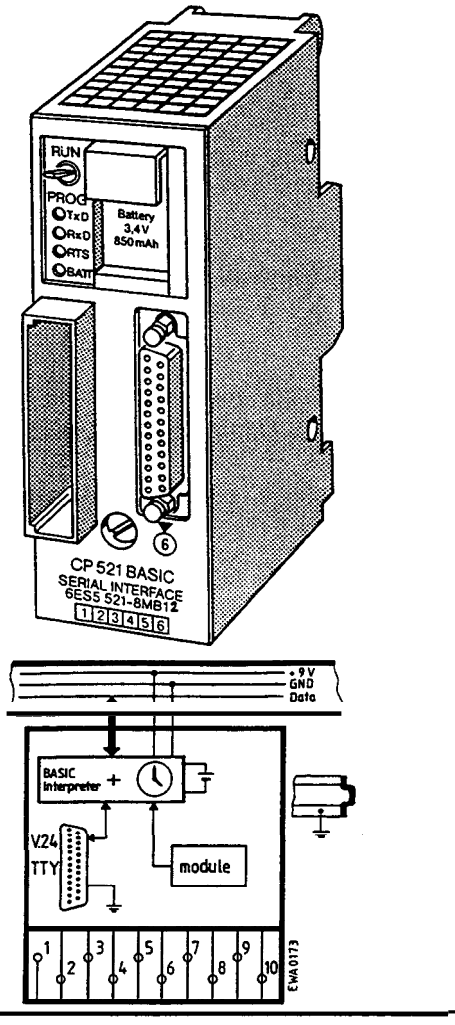
The EN 60204-1 standard (Safety of Machinery, Electrical Equipment of Machines, Part 1, Specification for General Requirements) applies for the electrical equipment of machinery.

The table below is designed to help you with the declaration of conformity and to show which criteria apply to SIMATIC according to EN 60204-1 (as at June 1993)

EN 60204-1	Subject/Criterion	Remarks
Paragraph 4	General requirements	Requirements are met in the devices are mounted/installed in accordance with the installation guidelines. Please observe the explanations in "Notes on CE Marking of SIMATIC S5".
Paragraph 11.2	Digital input/output interfaces	Requirements are met.
Paragraph 12.3	Programmable equipment	Requirements are met if the devices for protection of memory contents against change by unauthorized persons are installed in locked cabinets.
Paragraph 20.4	Voltage tests	Requirements are met.

2.3 Technical Specifications

Please refer to the manual for **your programmable controller for information on climatic, mechanical and electromagnetic conditions and ratings.**



Electrical isolation		TTY signals are isolatec
Memory submodule		EPROM/EEPROM/RAM
Bidirectional interface		RS 232 C (V.24)/TTY passive (active)
Transmission mode:		Asynchronous 10-bit character frame 1 l-bit character frame
Baud rate		110 to 9600 baud
Unidirectional interface		RS 232 C (V.24)
Permissible cable length -TTY		Voltage drop Receiver (typical): 1.5 V Sender (typical): 0.9 V
- RS 232 C (V.24)		15 m
Quartz frequency		14.7456 MHz
Battery failure indicator (yellow LED)		Yes
Lithium ½AA backup battery		3.6 V/850 mAh
Backup time		At least 1 year
Type of protection		IP 20
Permissible ambient temperature - Horizontal installation - Vertical installation		 O“Cto60“C o‘Cto40“c
Relative humidity		15% to 95%
Power consumption from +9 V (CPU)	Typ.	180 mA
Power loss (module)	Typ.	1.6 W
Weight	Approx.	500 g (1 [b.)

2.4 Memory Submodules

A memory submodule is required for storing BASIC programs or parameter initialization data.
There are three types of memory submodules:

Table 2-1. Erasing and Storing Data on Memory Submodules

Type of Submodule	Erase with	Store Programs with
EPROM	UV eraser	PG
EEPROM	PG/CP 521 BASIC	PG/CP 521 BASIC
RAM	PG/CP 521 BASIC	CP 521 BASIC

Table 2-2. Overview of Permissible Memory Submodules

Type of Submodule	Submodule Identifier	Capacity
EPROM	6ES5 375 - 1LA15	1 x 8 Kbytes
EPROM	6ES5 375 - 1LA21	2 x 8 Kbytes
EPROM	6ES5 375 - 1LA41	2 x 16 Kbytes
EEPROM	6ES5 375 - 0LC11	1 x 2 Kbytes
EEPROM	6ES5 375 - 0LC31	1 x 8 Kbytes
EEPROM	6ES5 375 - 0LC41	2 x 8 Kbytes
RAM	6ES5 375 - 0LD11	1 x 8 Kbytes
RAM	6ES5 375 - 0LD21	2 x 8 Kbytes
RAM	6ES5 375 - 0LD31	2 x 16 Kbytes

NOTE:

The memory submodule must never be removed while the CPU is in RUN mode.

Table 2-3. Overview of Permissible Memory Submodules that Can Still be Used

Type of Submodule	Submodule Identifier	Capacity
EPROM	6ES5 375 - 0LA15 or 11	1 x 8 Kbytes
EPROM	6ES5 375 - 0LA21	2 x 8 Kbytes
EPROM	6ES5 375 - 0LA41	2 x 16 Kbytes

2.5 Serial Interfaces

The CP 521 BASIC module is equipped with a bidirectional RS232C (V.24)/TTY interface (programmable) and a unidirectional RS232C (V.24) interface (e.g. for connecting a printer). The TTY interface is designed as passive interface, but can also be operated as active interface by supplying +24 V over the front connector. There is no electrical isolation when the TTY interface is operated as ***active*** interface. The distance between the CP 521 BASIC and the device connected to it over this interface may be up to 1000 m.
The following I/O devices an be connected to the CP 521 BASIC's serial interfaces:

- Programmer (PG)
- Keyboard
- Terminal
- Printer with RS232C (V.24) interface
- Printer with active TTY interface

NOTE:

It is not possible to use the unidirectional RS232C (V.24) interface when using the TTY interface.

2.6 Integral Real-Time Clock

The module is equipped with an integral real-time clock which can be battery-backed when the module is off power.
The real-time clock can be set or read out over the CPU using a programmer or in the BASIC program.
The clock supports the following functions:

- Seconds
- Minutes
- Hours (12/24-hr mode)
- Day
- Day-of-the-week
- Month
- Year (leap-years are taken into account)

2.7 Backup Battery

The module's backup battery backs up the real-time clock and RAM when the programmable controller is off power.
When a power failure occurs or when the programmable controller is switched off, the clock data is retained only when the module is equipped with a battery.
The battery should be inserted or changed while the programmmable controller is on. If the controller is off when the battery is changed, the clock would have to be reset when the controller is switched back on.
A new lithium battery has a service life of at least one year.

CAUTION:

Lithium batteries cannot be charged. Any attempt to do so could cause an explosion! Old batteries should always be taken to a special refuse disposal plant.

2.8 LEDs

The module is equipped with the following LEDs:

- One green Transmit LED
- One green Receive LED
- One green Ready To Send LED (RTS)
- One yellow battery failure LED (BATTERY OFF/LOW)

2.9 List of Accessories and Order Numbers

Memory submodules

EPROM submodule, 1 x 8 Kbytes	6ES5 375-1LA15
EPROM submodule, 2 x 8 Kbytes	6ES5 375-1LA21
EPROM submodule, 2 x 16 Kbytes	6ES5 375-1LA41
EPROM submodule, 1 x 2 Kbytes	6ES5 375-0LC11
EPROM submodule, 1 x 8 Kbytes	6ES5 375-0LC31
EPROM submodule, 2 x 8 Kbytes	6ES5 375-0LC41
EPROM submodule, 1 x 8 Kbytes	6ES5 375-0LD11
EPROM submodule, 2 x 8 Kbytes	6ES5 375-0LD21
EPROM submodule, 2 x 16 Kbytes	6ES5 375-0LD31

Printers

see S5-100U Catalog	ST 52.1

Programmers

see Programmer Catalog	ST 59

Backup battery

AA lithium battery, 3.6 V/850 mAh	6ES5 980-0MA11

3 Installation Guidelines

3.1 Installation . . . . . 3 .- 1

3.2 Dimension Diagram . . . . . 3 - 2

3.3 Wiring of Programmable Controllers for EMC . . . . . 3 - 3

3.3.1 Routing of Cables . . . . . 3. - 3

3.3.2 Equipotential Bonding . . . . . 3 - 6

3.3.3 Shielding of Cables and Lines . . . . . 3 - 8

3.4 Further Notes on System Configuration and Installation 3 - 10

3.5 I/O Interfaces . . . . . 3 .- 11

3.6 General Examples . . . . . 3. - 12

Figures

3-1. Setting the Coding Element . . . . . 3 - 1
3-2. Dimension Diagram of the CP 521 BASIC Module . . . . . 3 - 2
3-3. Routing of Equipotential Bonding Conductor and Signal Line . . . . . 3 - 7
3-4. Examples of Securing Shielded Lines with Cable Clamps . . . . . 3 - 9
3-5. Connection Diagram: CP 521 BASIC - Terminal (RS232C (V.24) Interface with HW Handshake) . . . . . 3 - 12
3-6. Connection Diagram: CP 521 BASIC - CP 521 BASIC, CP 521 BASIC - Printer (RS232C (V.24) Interface) . . . . . 3 - 13
3-7. Connection Diagram: CP 521 BASIC - PG 7xx, CP 521 BASIC - Printer (RS232C (V.24) Interface) . . . . . 3 - 14
3-8. Connection Diagram: CP 521 BASIC (TTY Active) - PG 685/PG 635 (TTY Passive) . . . . . 3 - 15
3-9. Connection Diagram: CP 521 BASIC (TTY Passive) - PG 685/PG 635 (TTY Active) . . . . . 3 - 16
3-10. Connection Diagram: CP 521 BASIC (TTY Passive) - PG 750/PG 730 (TTY Active) . . . . . 3 - 17

Tables

3-1. Rules for Laying of Line Combinations . . . . . 3 - 4
3-2. Pin Assignments for the 25-Pin D Sub Socket Connector 3 - 11

3 Installation Guidelines

3.1 Installation

You must note the following when installing and removing the CP 521 BASIC module:

- **The module may be inserted or removed only when both module and programmable controller are off power.**
- **The memory submodule maybe inserted or removed only when the module is off power.**
- **The CP 521 BASIC module maybe connected to or disconnected from the 1/0 device (D sub socket connector) only when the data transfer between CP 521 BASIC and I/O device has been completed.**
- **The module may be snap-mounted only into slots O to 7 of a bus unit (slot 7 may not be used for CPU 102 (6ES 5 102-8MAO1 up to revision level 5) or CPU 103 (6ES5 103-8 MAO1, revision level 1).**
- **The number 6 must be set on the bus unit's coding element before installing the CP 521 BASIC module.**

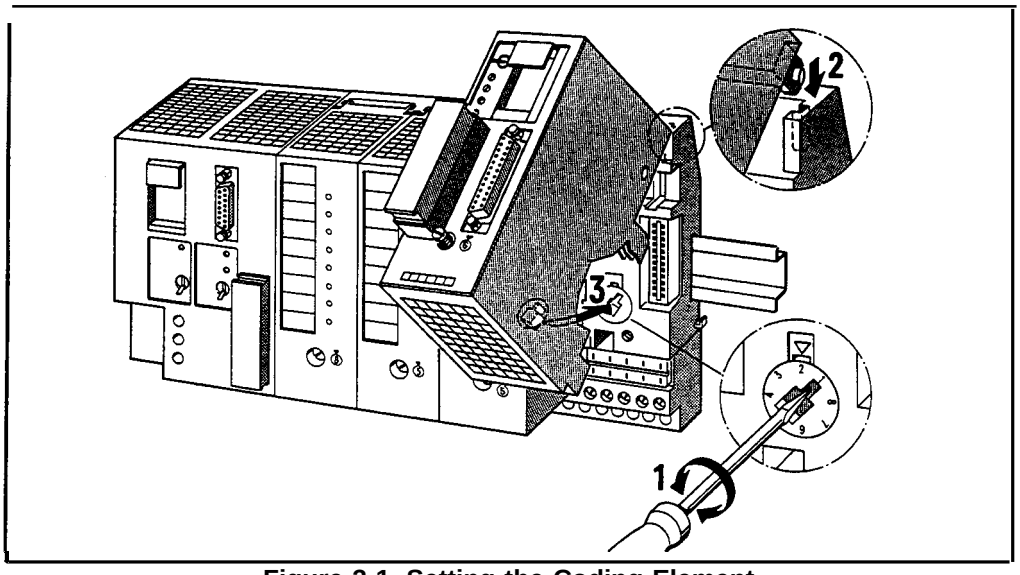


Figure 3-1. Setting the Coding Element

Securing the module:

1. **Hook the module onto the top of the bus unit and swing it down onto the bus unit.**
2. **Press the module firmly into place and screw it onto the bus unit.**

3.2 Dimension Diagram

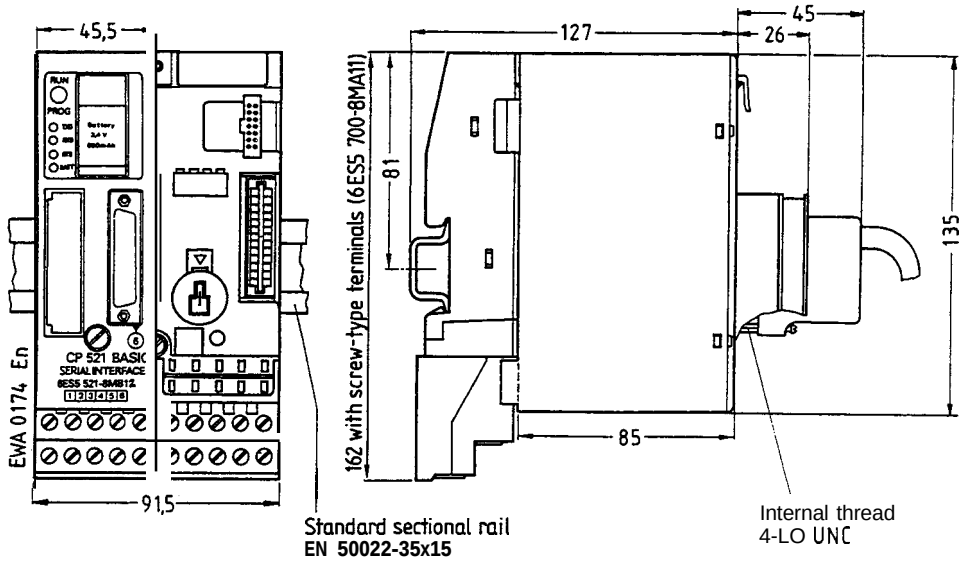


Figure 3-2. Dimension Diagram of the CP 521 BASIC Module

3.3 Wiring of Programmable Controllers for EMC

The following section describes:

- Routing of cables within and outside cabinets
- Equipotential bonding between devices
- Connection of cable shields

3.3.1 Routing of Cables

This section covers the routing of bus, signal and supply lines. The purpose of cable routing is to suppress crosstalk between cables laid in parallel.

Routing of Cables Within and Outside Cabinets

For electromagnetically compatible routing of cables and lines, it is advisable to subdivide the lines into the following line groups and lay the groups separately.

Group A: Shielded bus and data lines (for programmer, OP, SINEC L1, SINEC L2, printer, etc.)
Shielded analog lines
Unshielded lines for DC voltage 60 V
Unshielded lines for AC voltage 25 V

Group B: Unshielded lines for DC voltage >60 V and 400 V
Unshielded lines for AC voltage >25 V and 400 V

Group C: Unshielded lines for DC and AC voltage>400 V

Group D: Lines for SINEC H1

From the combination of individual groups in the following table, you can read off the conditions for laying the line groups.

Table 3-1 Rules for Laying of Line Combinations

	Group A	Group B	Group C	Group D
Group A				
Group B				
Group C				
Group D				

Legend for the table:

Lines can be laid in common bundles or cable ducts.
Lines must be laid in separate bundles or cable ducts (without minimum clearance).
Lines within cabinets must be laid in separate bundles or cable ducts; outside the cabinets but within buildings, they must be laid over separate cable routes with a clearance of at least 10 cm (3.93 in.).
Lines must be laid in separate bundles or cable ducts with a clearance of at least 50 cm (1.64 ft.).

Routing of Cables Outside Buildings

Outside buildings, lay the lines on metal cable trays if possible. Provide the joints between cable trays with an electrical connection and ground the cable trays.

When laying lines outside buildings, you must observe the valid lightning protection and grounding measures. The following applies in general:

Lightning Protection

Where cables and lines for SIMATIC S5 controllers are to be laid outside buildings, you must apply measures for internal and external lightning protection.

Outside the buildings, lay your lines either

- in metal conduits grounded at both ends,
  or
- in concreted cable ducts with continuously connected reinforcement.

Protect the signal lines from overvoltages by means of

- varistors
  or
- inert gas-filled surge diverters.

Fit these protective devices at the cable entry into the building.

> **Note**
>
> Lightning protection measures always require an individual assessment of the entire installation. For clarification, please consult your local Siemens Office or a company specializing in lightning protection, such as Messrs. Dehn und Söhne in Neumarkt both Germany.

Equipotential Bonding

Ensure adequate equipotential bonding between the connected equipment (see section 3.3.2).

3.3.2 Equipotential Bonding

Between separate sections of an installation, potential differences can develop if

- programmable controllers and I/O devices are connected via non-floating links,
  or
- cable shields are connected at both ends and are grounded at different parts of the system.

Different AC supplies, for example, can cause potential differences. These differences must be reduced by installing equipotential bonding conductors to ensure functioning of the electronic components.

The following points must be observed for equipotential bonding:

- The lower the impedance of the equipotential bonding conductor, the greater is the effectiveness of equipotential bonding.
- Where shielded signal lines are laid between the relevant sections of the system and connected at both ends to the ground/protective conductor, the impedance of the additional equipotential bonding conductor must not exceed 10 % of the shield impedance.
- The cross-section of the equipotential bonding conductor must be rated for the maximum circulating current. The following cross-sections of copper have proved to be satisfactory in practice:
  - 16 mm^2 for equipotential bonding conductors of up to 200 m (656 ft.) in length
  - 25 mm^2 for equipotential bonding conductors of more than 200 m (656 ft.) in length.
- Use copper or zinc-plated steel for equipotential bonding conductors. They must be given a large-area connection to the ground/protective conductor and protect it from corrosion.
- The equipotential bonding conductor should be laid so that the smallest possible areas are enclosed between the equipotential bonding conductor and signal lines (see section 3.3).

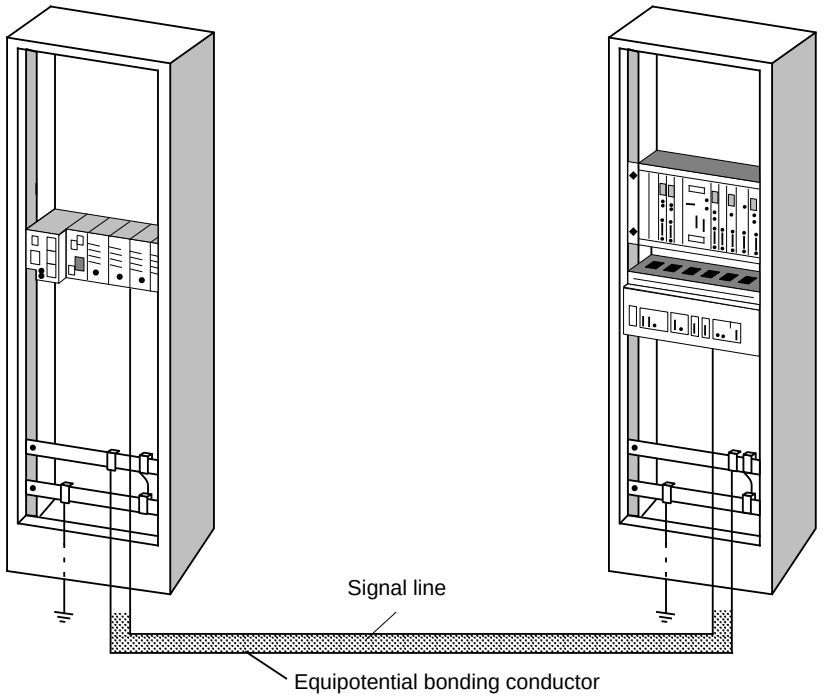


Figure 3-3. Routing of Equipotential Bonding Conductor and Signal Line

3.3.3 Shielding of Cables and Lines

Shielding is a method of attenuating magnetic, electrical or electromagnetic interference fields.

Interference currents on cable shields are passed to ground via the shield bar which is electrically connected to the housing. A low-impedance connection to the protective conductor is particularly important so that these interference currents themselves do not become an interference source.

Where possible, only use lines with a braided shield. The coverage density of the shield should be more than 80 %. Avoid lines with a foil shield because the foil can be very easily damaged by tensile strain and compression during fitting; this results in reduced effectiveness of the shield.

As a rule, line shields should always be connected at both ends. This is the only way to achieve a good degree of interference suppression in the higher frequency region.

Only in exceptional cases should the shield be connected at one end only, as this only achieves attenuation of the low frequencies. Single-ended shield connection may be more advantageous when:

- an equipotential bonding conductor cannot be laid;
- analog signals (of a few mV or mA) are to be transmitted;
- foil (static) shields are used.

With data lines for serial communication, always use metal or metallized connectors. Secure the shield of the data line to the connector case. Do **not** connect the shield to PIN 1 of the connector.

For stationary operation, it is advisable to fully strip the insulation from the shielded cable and connect it to the shield/protective conductor bar.

Note

In the event of potential differences between ground points, a circulating current may flow through the shield connected at both ends. In this case, install an additional equipotential bonding conductor (see section 3.3.2).

Please observe the following points when connecting the shield:

- Use metal cable clamps for securing the braided shield. The clamps must enclose the shield over a large area and provide a good contact (see Figure 3-4).
- Connect the shield to a shield bar immediately after the cable entry into the cabinet. Route the shield as far as the module but do not connect it there again.

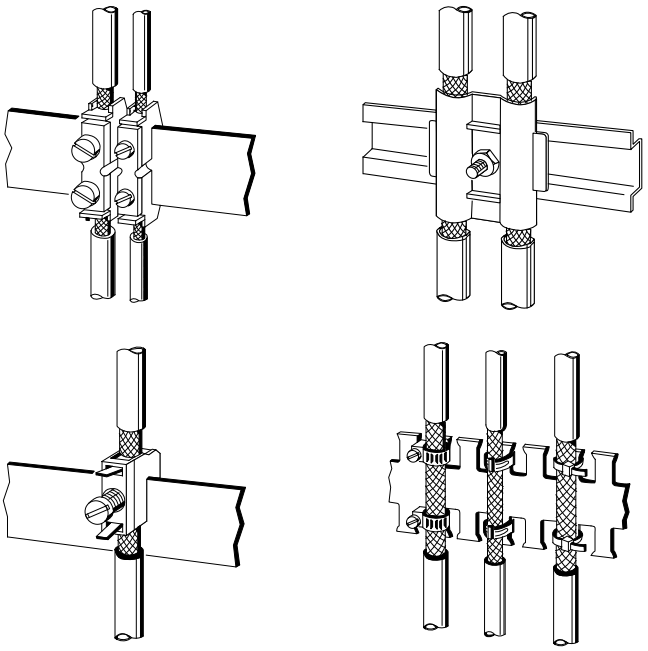


Figure 3-4. Examples of Securing Shielded Lines with Cable Clamps

3.4 Further Notes on System Configuration and Installation

Since the module is normally used as a component part of a larger system or plant, these notes are aimed at the hazard-free integration of the product in its environment.

The following are notes to be observed for the installation and startup of the product:

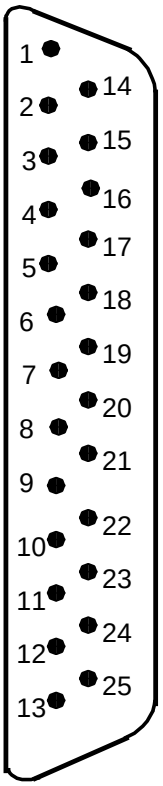
Warning

- Strictly follow the safety and accident prevention rules that apply in each particular case.
- In the case of equipment with a permanent power connection not provided with an isolating switch and/or fuses which disconnect all poles, a suitable isolating switch or fuses must be provided in the building wiring system (distribution board). Furthermore, the equipment must be connected to a protective ground (PE) conductor.
- In the case of equipment operated on the system voltage, make sure that the rated voltage range set coincides with the local system voltage before taking the equipment into operation.
- In the case of equipment operating on 24 V DC, make sure that the proper electrical isolation is provided between the mains supply and the 24 V supply. Use only power supply units to DIN VDE 0551 / EN 60742 und DIN VDE 0160.
- Fluctuations or deviations of the power supply voltage from the rated value should not exceed the tolerances specified in the technical specifications, otherwise function failures or dangerous conditions can occur in the electronic modules/equipment.
- Suitable measures must be taken to make sure that programs that are interrupted by a voltage dip or power supply failure resume proper operation when the power supply is restored. Care must be taken to ensure that dangerous operating conditions do not occur even momentarily. If necessary, the equipment must be forced into the ”emergency off” state.
- Emergency tripping devices in accordance with EN 60204/IEC 204 (VDE 0113) must be effective in all operating modes of the automation equipment. Resetting the emergency off device must not result in uncontrolled or undefined restart of the equipment.
- Connecting cables and signal cables must be installed in such a way that inductive and capacitive interference has no adverse effects on the automation functions.
- Automation equipment and operator controls must be installed in such a way that they are adequately protected against unintentional operation.
- To prevent wire breaks on the signal side leading to undefined states in the automation equipment, the relevant hardware and software precautions must be taken in the case of I/O connections.

3.5 I/O Interfaces

The module is equipped with one bidirectional current or voltage interface (TTY/RS232C (V.24)) and one unidirectional voltage interface (RS232C (V.24)). The bidirectional interface can be initialized as TTY or RS232C (V.24) interface. All circuits for both the bidirectional and unidirectional interfaces are routed to a 25-pin D sub socket. The V.24 interface signals conform to the RS232C standard.

Table 3-2. Pin Assignments for the 25-Pin D Sub Socket Connector

Connector	Pin No.	Signal Name	Description
	1	-	
	2	TxD	Transmit Data (V.24)
	3	RxD	Receive Data (V.24)
	4	RTS	Request to Send (V.24)
	5	CTS	Clear to Send (V.24)
	6	DSR	Data Set Ready (V.24)
	7	GND	Signal ground
	8	-	(V.24/RS232C)
	9	RxD+	
	10	RxD-	TTY receive circuit+
	11	-	TTY receive circuit -
	12	-	
	13	P24	
	14	LPO	+24 V for active TTY
	15	BUSY	Line Printer Out
	16		BUSY signal
	17	I20B	
	18	TxD+	Current source TTY *
	19	I20A	TTY transmit circuit+
	20	DTR	Current source TTY*
	21	TxD-	Data Terminal Ready
	22	RI/T	TTY transmit circuit -
	23	-	Request In/Timing
	24	-	
	25	PM1	Test loop enabled

***** If +24 V supplied over pin 13

3.6 General Examples

This section presents a number of examples for the module's communications/ printer interface.

CP 521 BASIC to terminal (RS232C (V.24) interface with hardware handshake)

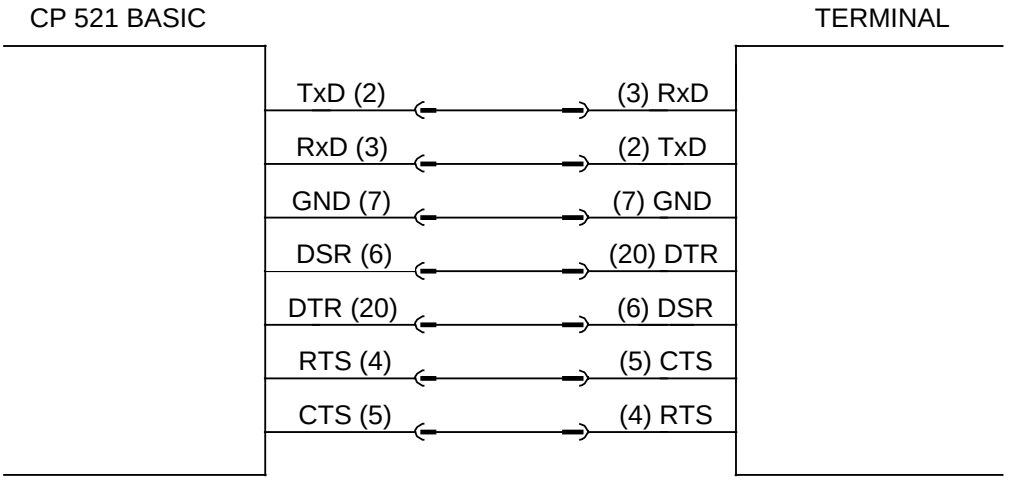


Figure 3-5. Connection Diagram: CP 521 BASIC - Terminal (RS232C (V.24) Interface with HW Handshake)

CP 521 BASIC to CP 521 BASIC,
CP 521 BASIC to printer (RS232C (V.24) interface with hardware handshake)

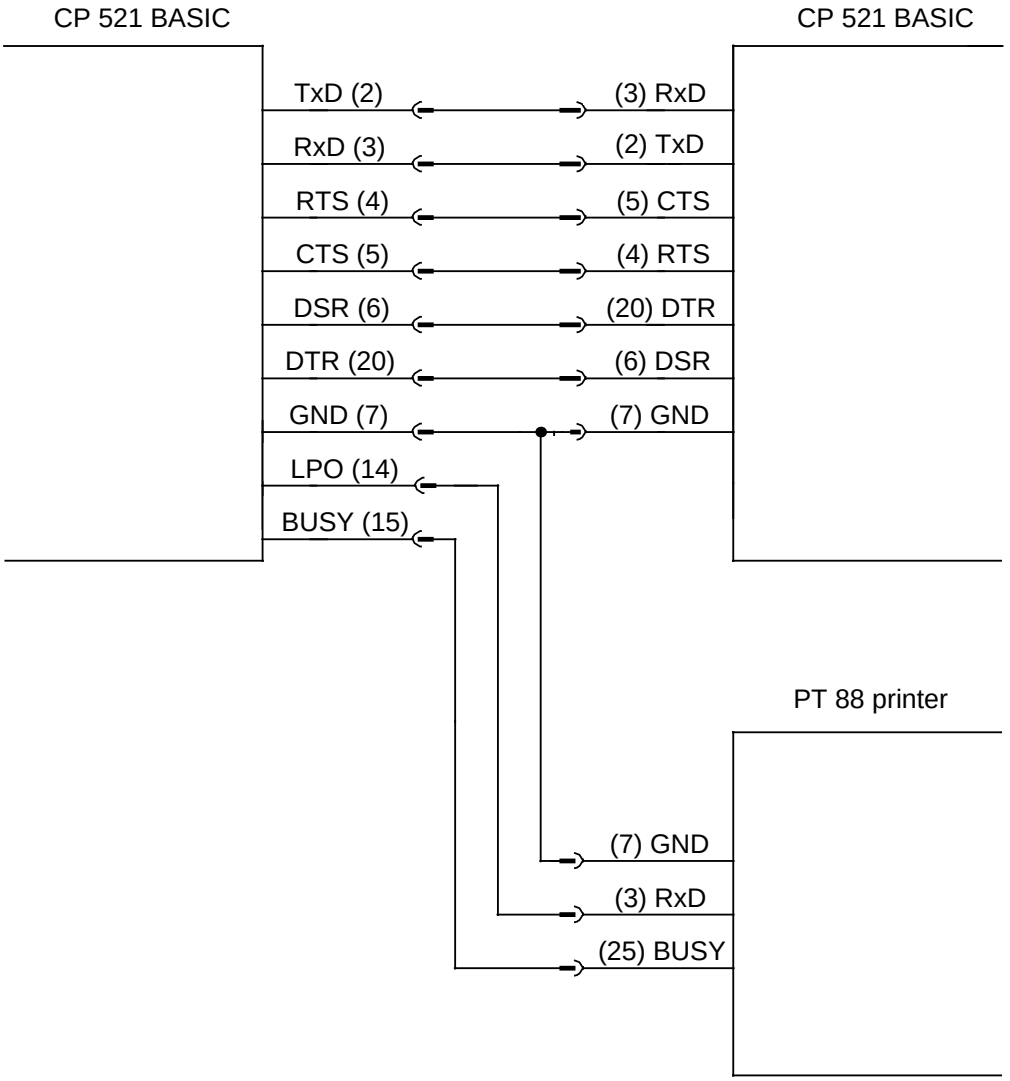


Figure 3-6. Connection Diagram: CP 521 BASIC - CP 521 BASIC, CP 521 BASIC - Printer (RS232C (V.24) Interface)

CP 521 BASIC to PG 7xx,
CP 521 BASIC to PT 88 printer (RS232C (V.24) Interface)

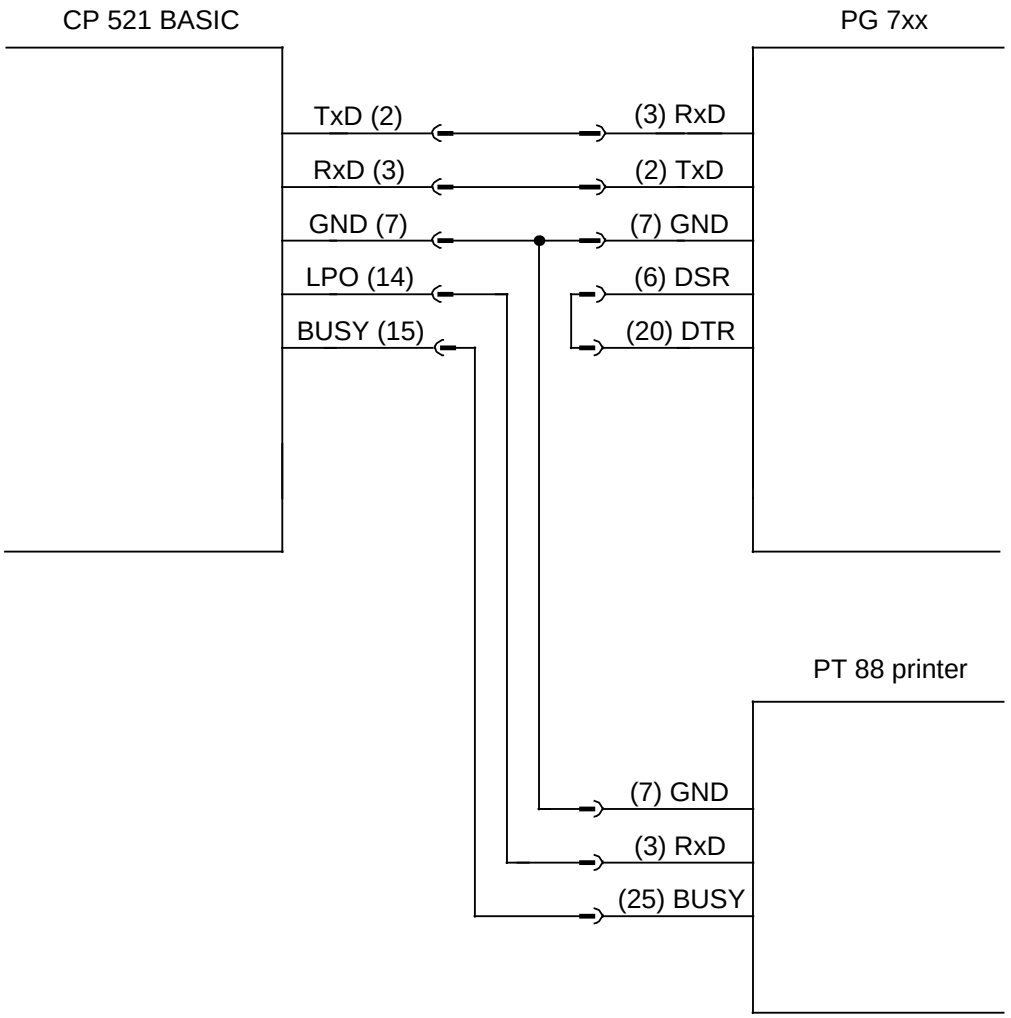


Figure 3-7. Connection Diagram: CP 521 BASIC - PG 7xx, CP 521 BASIC - Printer (RS232C (V.24) Interface)

CP 521 BASIC (TTY active) to PG 685/PG 635 (TTY passive)

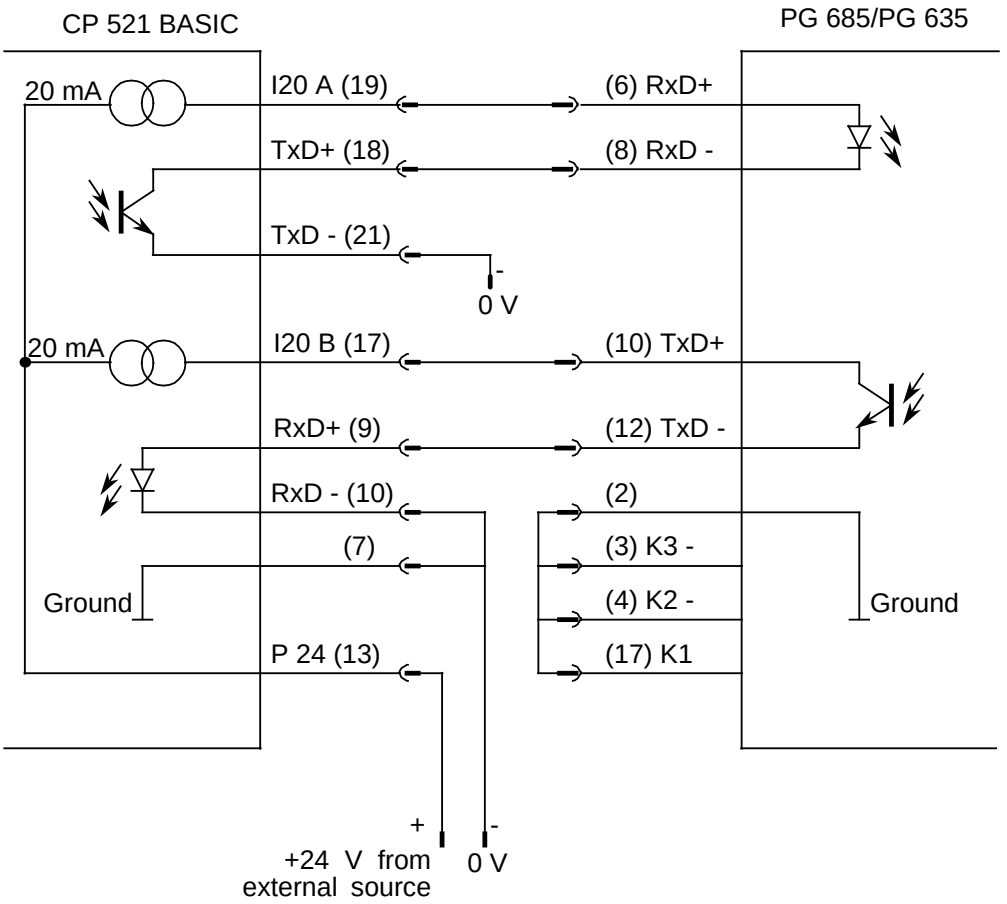


Figure 3-8. Connection Diagram: CP 521 BASIC (TTY Active) - PG 685/PG 635 (TTY Passive)

The figure above includes a schematic of the current flow.

CP 521 BASIC (TTY passive) to PG 685/PG 635 (TTY active)

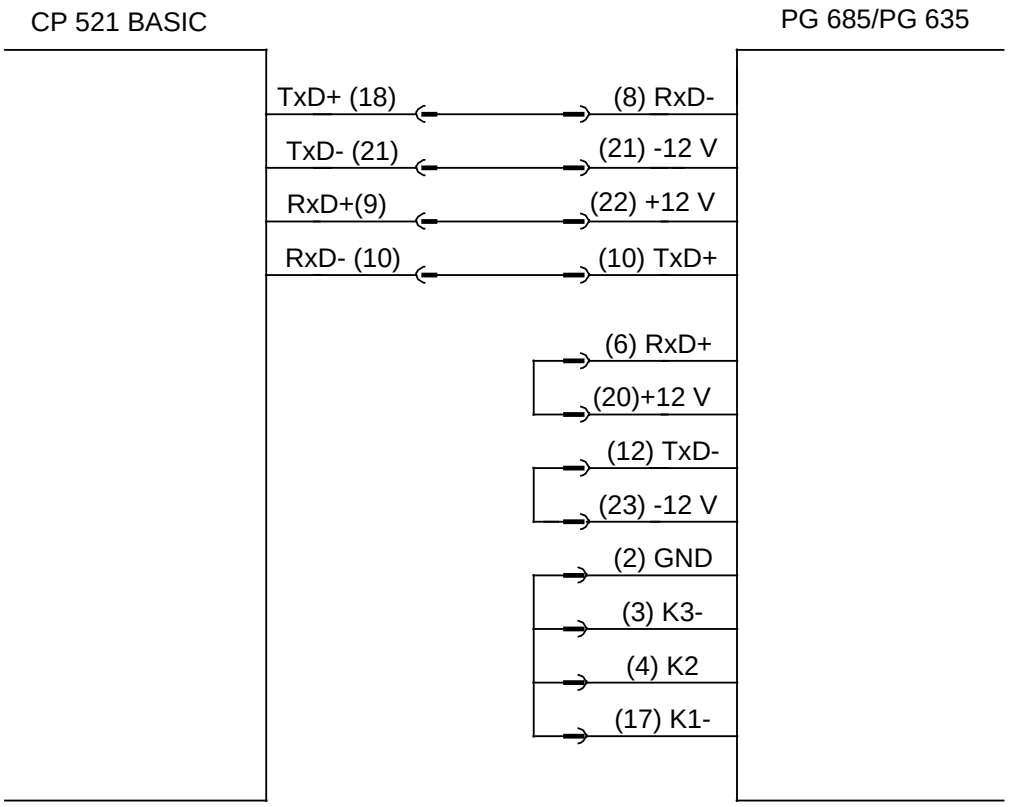


Figure 3-9. Connection Diagram: CP 521 BASIC (TTY Passive) - PG 685/PG 635 (TTY Active)

A baud rate of 9600 can be set by connecting pins (2), (3), (4) and (17) on the PG side.

CP 521 BASIC (TTY passive) to PG 7xx (TTY active)

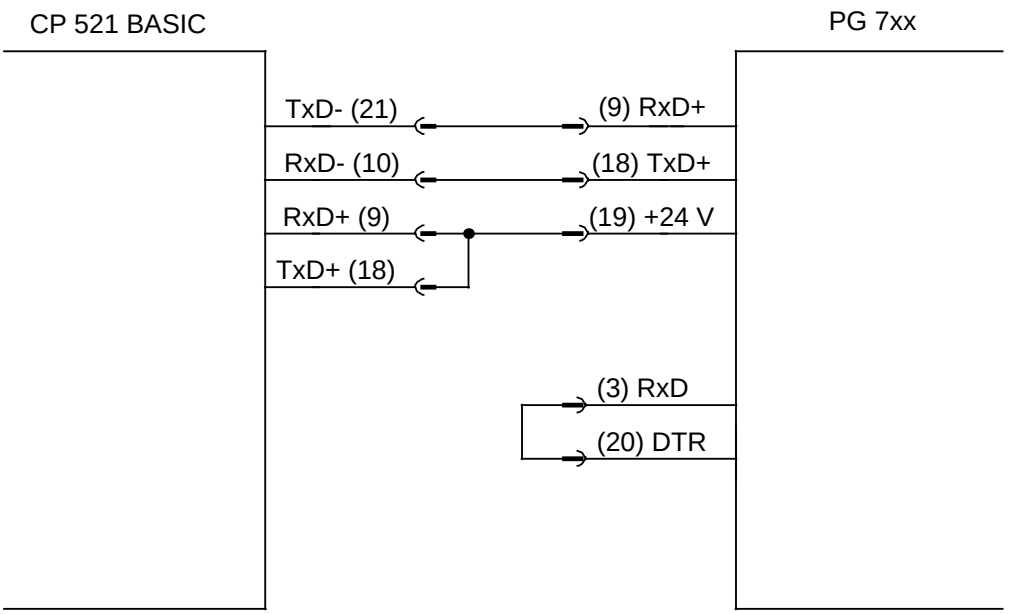


Figure 3-10. Connection Diagram: CP 521 BASIC (TTY Passive) - PG 750/PG 730 (TTY Active)

4 Principle of Operation and Addressing

4.1 General Principle of Operation . . . . . 4 - 1

4.2 Fundamentals of Data Interchange . . . . . 4 - 2

4.3 Restart Characteristics . . . . . 4 - 3

4.3.1 Battery Test . . . . . 4 - 4

4.3.2 Clock Test . . . . . 4 - 4

4.4 Mode Switches . . . . . 4 - 5

4.5 Addressing . . . . . 4 - 6

Figures

4-1. Schematic of the CP 521 BASIC . . . . . 4 - 1
4-2. System Environment . . . . . 4 - 2

Tables

4-1. Clock Test Errors . . . . . 4 - 4
4-2. Response to Actuation of the Mode Switches . . . . . 4 - 5
4-3. Slot Addresses . . . . . 4 - 6

4 Principle of Operation and Addressing

4.1 General Principle of Operation

The CP 521 BASIC handles data interchange with an I/O device autonomously. The CPU always takes the initiative for a data transfer between itself and the CP 521 BASIC by forwarding a request to the CP 521 BASIC.

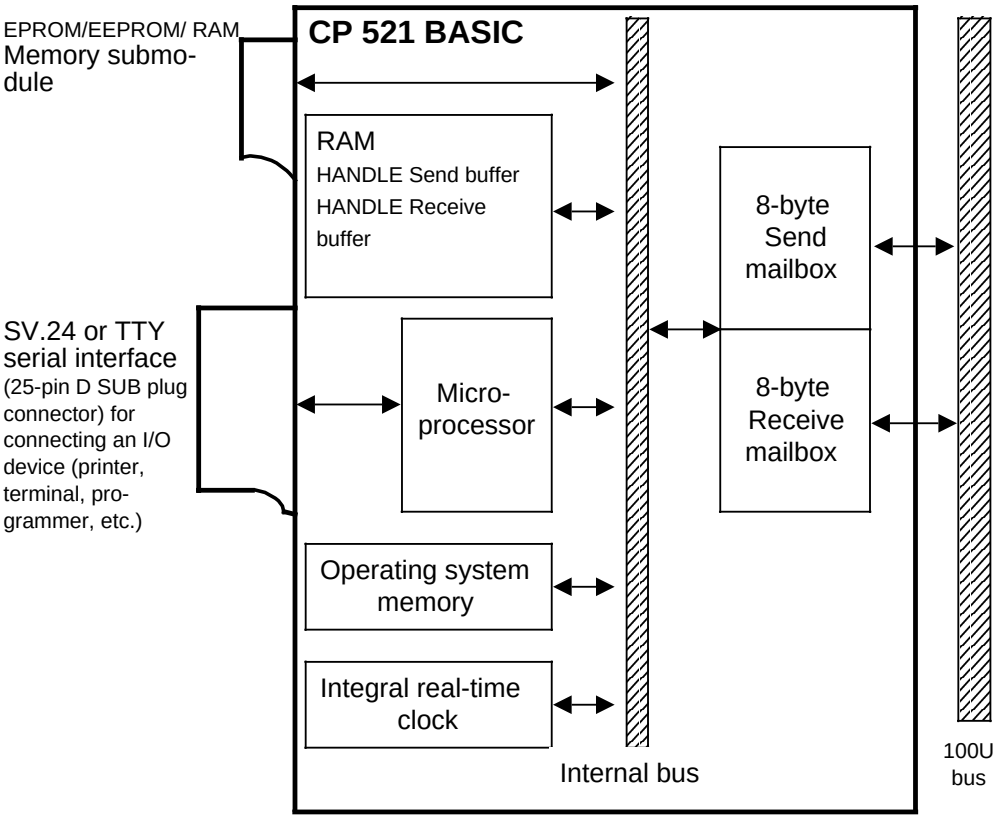


Figure 4-1. Schematic of the CP 521 BASIC

4.2 Fundamentals of Data Interchange

The CPU must initiate every data interchange between itself and the CP 521 BASIC module by forwarding a request to the CP 521 BASIC. The CPU controls the CP 521 BASIC's data interchanges.

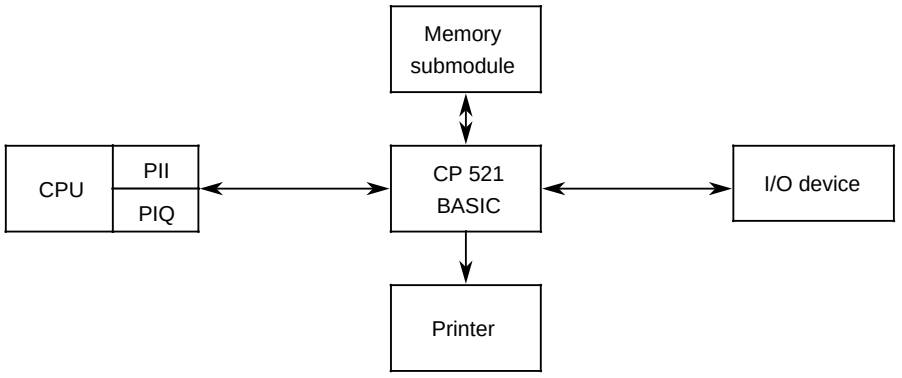


Figure 4-2. System Environment

The CP 521 BASIC is equipped with four interfaces:

Interface to the 100U bus for communicating with the CPU

- The control program forwards data to the module over the 100U bus, and evaluates data from the module.
- The 100U bus forwards 8 bytes from the CPU to the CP 521 BASIC and 8 bytes from the CP 521 BASIC to the CPU.

Receptacle for submodule

Bidirectional TTY and RS232C (V.24) interface for interchanging data with an I/O device or terminal.

Unidirectional RS232C (V.24) interface for connecting a printer.

Permissible I/O devices:
- Programmer
- Keyboard
- Terminal
- Printer with RS232C (V.24) interface
- Other I/O devices with serial interface

The memory submodule is needed for storing BASIC programs. For a list of permissible memory submodules, see 2.7 (List of Accessories).

4.3 Restart Characteristics

The CP 521 BASIC executes a restart on power return (POWER ON).

A restart consists of the following:
- Initialization of the serial interfaces
- Battery test (4.3.1)
- Clock test (4.3.2)
- Scanning of the PROG/RUN switch (mode selector switch)

For the most part, the CP 521 BASIC's restart characteristics are parameter-controlled (5.2):
- RAM/clock failure or battery failure
- The module's mode switch setting (PROG/RUN)
- Parameters for the last ”PROG” instruction (Table 5.1).

4.3.1 Battery Test

When the module is off power, the clock and the RAM are backed by a battery. The battery is tested on every restart. In the event of a battery failure, the data and/or BASIC programs in RAM are lost. The BATTERY LOW LED is set if the battery voltage falls below the required backup voltage during operation or during a restart.

4.3.2 Clock Test

This part of the restart routine tests the module's hardware clock. The clock is set when the battery fails while the PLC is off power. The clock defaults to ”01.01.90 00.00”. If the clock fails or reverts to the default, the appropriate error code is entered in byte 0 (module status) and can be subsequently evaluated in the user program or via a programmer.

Table 4-1. Clock Test Errors

Error code in byte 0	Error	Corrective measures
$1X_H$	Clock defect	Replace module
$2X_H$	Clock set Clock reverts to default	Forward the current clock data to the CP 521 BASIC using the programmer's ”FORCE VAR” function.

X : Can assume different values

N O T E :

If the clock or the battery fails, it must be assumed that the data in RAM is no longer correct. For this reason, all of the CP 521 BASIC's system variables are set to their default values and the RAM is cleared when a battery or clock failure occurs while the module is off power.

4.4 Mode Switches

Following POWER ON, the CP 521 BASIC's operating mode depends on the following mode selector switches:

- STOP/RUN - swtich on the CPU
- PROG/RUN - switch on the CP 521 BASIC.

Table 4-2. Response to Actuation of the Mode Switches

Mode switch settings	Mode	Status
CPU: RUN STOP (RUN) CP 521 BASIC: RUN or PROG	Execution of the BASIC program is not interrupted	- 100U bus inactive - No data transfer between CPU and CP 521 BASIC* - Clock data is not updated
CPU: RUN CP 521 BASIC: RUN PROG (RUN)	Execution of the BASIC program is not interrupted	- 100U bus active - Data transfer between CPU and CP 521 BASIC - Clock data is continually updated

* The ”S5 bus inactive” status can be read out with the SSTAT command (5.9.3)

NOTE:

The CP 521 BASIC's mode selector switch is scanned in the restart routine only. Depending on its setting, a preselected program can be started or the module can restart in the Command mode.

4.5 Addressing

The CPU addresses the CP 521 BASIC in the address area of the analog channels. The module has an address area of 8 bytes for input and 8 bytes for output. Input data and output data are referenced over the same address area.
Note the following carefully:

- The module may be inserted in slots 0 to 7 only.
- The address space extends from byte 64 to byte 127.
- Eight bytes are reserved in the CPU's process input image (PII) and eight in the CPU's process output image (PIQ) per slot.

The process input image (PII) is used to store data forwarded from the module to the CPU.
Information forwarded from the CPU to the module is entered under the same addresses in the process output image (PIQ).

Table 4-3. Slot Addresses

Slot	0	1	2	3	4	5	6	7
PII/PIQ addresses	**64** ... 71	**72** ... 79	**80** ... 87	**88** ... 95	**96** ... 103	**104** ... 111	**112** ... 119	**120** ... 127
Start address of a slot								

The eight bytes (0 to 7) reserved for each slot serve a specific purpose. The byte numbers given in this manual always refer to the start address of the relevant slot. In your control program, you must therefore add the start address of the slot in which you inserted the module to the byte numbers given in the manual.

Example: Module is in slot 3: Byte 2 has the address 90
Module is in slot 4: Byte 2 has the address 98

Byte 0 (”Request”) of the PIQ determines the meaning of bytes 1 to 7 in the PIQ.

5 BASIC Interpreter

5.1 Introduction . . . . . 5 - 1

5.2 Restart Characteristics . . . . . 5 - 2

5.3 Programming . . . . . 5 - 6
5.3.1 Command Mode . . . . . 5 - 6
5.3.2 RUN Mode . . . . . 5 - 6

5.4 COM Software . . . . . 5 - 7
5.4.1 Terminal Emulation . . . . . 5 - 11
5.4.2 Save BASIC Programs. . . . . 5 - 13
5.4.3 Loading BASIC Programs . . . . . 5 - 15
5.4.4 Displaying the Contents of Memory and Files . . . . . 5 - 18

5.5 The BASIC Interpreter . . . . . 5 - 20
5.5.1 Statements/Format Statements . . . . . 5 - 20
5.5.2 Data Formats . . . . . 5 - 21
5.5.3 Operators . . . . . 5 - 22
5.5.4 Variables . . . . . 5 - 22
5.5.5 Expressions/Relational Expressions . . . . . 5 - 24
5.5.6 System Control Parameters . . . . . 5 - 24
5.5.7 Stacks . . . . . 5 - 25
5.5.8 Line Editor . . . . . 5 - 25

5.6 Command Descriptions . . . . . 5 - 26
5.6.1 RUN . . . . . 5 - 26
5.6.2 CONT . . . . . 5 - 27
5.6.3 LIST . . . . . 5 - 28
5.6.4 LIST# . . . . . 5 - 29
5.6.5 NEW . . . . . 5 - 30

5.7 Memory Commands . . . . . 5 - 31
5.7.1 RAM and ROM . . . . . 5 - 31
5.7.2 XFER . . . . . 5 - 33
5.7.3 PROG . . . . . 5 - 34
5.7.4 PROG1 and PROG2 . . . . . 5 - 36
5.7.5 ERASE . . . . . 5 - 37
5.7.6 REORG . . . . . 5 - 37

5.8 Statement Descriptions . . . . . 5 - 38
5.8.1 CLEAR . . . . . 5 - 38
5.8.2 CLEARI and CLEARS . . . . . 5 - 39
5.8.3 INIT . . . . . 5 - 40
5.8.4 INIT# . . . . . 5 - 42
5.8.5 BREAK . . . . . 5 - 44
5.8.6 DATA - READ - RESTORE . . . . . 5 - 45
5.8.7 DIM . . . . . 5 - 47
5.8.8 DO - UNTIL . . . . . 5 - 48
5.8.9 DO - WHILE . . . . . 5 - 49
5.8.10 END . . . . . 5 - 50
5.8.11 FOR - TO - {STEP} - NEXT . . . . . 5 - 51
5.8.12 GOSUB - RETURN . . . . . 5 - 53
5.8.13 GOTO . . . . . 5 - 55
5.8.14 ON GOTO and ON GOSUB . . . . . 5 - 56
5.8.15 IF - THEN - ELSE . . . . . 5 - 57
5.8.16 INPUT . . . . . 5 - 59
5.8.17 LET . . . . . 5 - 62
5.8.18 ONERR . . . . . 5 - 64
5.8.19 ONTIME . . . . . 5 - 66
5.8.20 CLOCK1 and CLOCK0 . . . . . 5 - 68
5.8.21 TIME . . . . . 5 - 70
5.8.22 PRINT . . . . . 5 - 71
5.8.23 PRINT# . . . . . 5 - 76
5.8.24 PH0., PH1., PH0.#, PH1.# . . . . . 5 - 77

5.8.25 PUSH . . . . . 5 - 78
5.8.26 POP . . . . . 5 - 80
5.8.27 REM . . . . . 5 - 82
5.8.28 RETI . . . . . 5 - 83
5.8.29 STOP . . . . . 5 - 84
5.8.30 RROM . . . . . 5 - 85

5.9 Special Statements and Functions for Data Interchange 5 - 86
5.9.1 SGET . . . . . 5 - 89
5.9.2 SPUT . . . . . 5 - 90
5.9.3 SSTAT . . . . . 5 - 90
5.9.4 RSTAT . . . . . 5 - 91
5.9.5 BUFREAD . . . . . 5 - 91
5.9.6 BUFWRITE . . . . . 5 - 92
5.9.7 SEND . . . . . 5 - 93

5.10 Special Statements and Functions for Data Interchange Over the Serial Interfaces . . . . . 5 - 94
5.10.1 HANDLE, HANDLE# . . . . . 5 - 95
5.10.2 HSSTAT . . . . . 5 - 96
5.10.3 BUFIN . . . . . 5 - 97
5.10.4 HRSTAT . . . . . 5 - 99
5.10.5 GET . . . . . 5 - 100
5.10.6 BUFOUT . . . . . 5 - 101

5.11 Arithmetic/Logical Operators and Expressions . . . . . 5 - 102
5.11.1 Dyadic Operators . . . . . 5 - 102
5.11.2 Unary Operators - General Use . . . . . 5 - 105
5.11.3 Unary Operators - Logarithm Functions . . . . . 5 - 107
5.11.4 Unary Operators - Trigonometric Functions . . . . . 5 - 108

5.11.5 Operator Priorities . . . . . 5. - 110
5.11.6 Relational Expressions . . . . . 5 - 111

5.12 String Operations . . . . . 5. - 112
5.12.1 STRING . . . . . 5. - 113
5.12.2 ”+” (Combining Two Strings) . . . . . 5 - 115
5.12.3 MID . . . . . 5. - 116
5.12.4 FIND . . . . . 5. - 117
5.12.5 SLEN . . . . . 5. - 118
5.12.6 String Operators ASC and CHR . . . . . 5 - 118

5.13 Functions and Statements for Date and Time. . . . . 5 - 123
5.13.1 SETCLOCK . . . . . 5. - 123
5.13.2 RDCLOCK . . . . . 5. - 124
5.13.3 SETDATE. . . . . 5. - 125
5.13.4 RDDATE . . . . . 5. - 126
5.13.5 T_ADJ . . . . . 5. - 127

5.14 Miscellaneous Functions and System Control Parameters 5 - 128
5.14.1 Special Function Operators . . . . . 5 - 128
5.14.2 System Control Parameters . . . . . 5 - 129

5.15 Error Messages and Special Features. . . . . 5 - 130
5.15.1 Error Messages . . . . . 5. - 130
5.15.2 Special Features . . . . . 5. - 134

Figures

5-1. The COM Software's Start Menu . . . . . 5 - 9
5-2. The COM ”Defaults” Menu . . . . . 5 - 10
5-3. The COM ”Terminal Emulation” Menu . . . . . 5 - 12
5-4. The COM ”Save” Menu . . . . . 5 - 14
5-5. The COM ”Load” Menu . . . . . 5 - 16
5-6. The COM ”Contents” Menu . . . . . 5 - 19
5-7. Data Interchange Between CPU, CP 521 BASIC and I/O Device . . . . . 5 . - 86

Tables

5-1. Restart Characteristics of the CP 521 BASIC with BASIC Program/Following Initialization . . . . . 5 - 3

5 BASIC Interpreter

5.1 Introduction

This section is designed as a reference for users who are familiar with general programming concepts, not as a textbook for learning general programming techniques or BASIC.

In Section 3.5 you will find everything you need to put the module hardware into operation, including answers to such questions as:

- How must my terminal be wired?
- How do I go about connecting a printer?

Section 5.2 covers start and restart characteristics and procedures up to the point of entering the first line of the BASIC program.

Sections 5.3 and 5.4 deal with the writing of programs and with the special features of the COM software for the CP 521 BASIC module.

Section 5.5 provides general information on the BASIC interpreter, including number notation and variables.

Sections 5.6 to 5.14 describe in detail all BASIC instructions and all auxiliary instructions for communications with the CPU and the I/Os. It also includes a brief discussion of variables and some additional details on the BASIC language.

Section 5.15 deals with error messages and their evaluation while writing programs on the CP 521 BASIC, and with the evaluation of errors by the STEP 5 program on the CPU.

Note
The phrase ”output to console” as used in Section 5 refers to output over the bidirectional interface.

5.2 Restart Characteristics

A distinction must be made between two different situations:
1. The module is being put into operation for the first time.
2. The module already contains a program/has already been initialized.

Putting the CP 521 BASIC into operation for the first time
When the CP 521 BASIC is switched on after connecting the serial input device, nothing immediately noticeable happens. The reason for this is that the module first initializes the hardware and then starts the Auto Baud routine. The user must press the ***space key*** or, when using the COM software, the ”Init” function key on the serial input device to set the baud rate and start the SIGN ON procedure, which displays ”CP 521*MSC BASIC Vx.x*” on the monitor screen. We would strongly recommend pressing the space key first so that you can begin working with CP 521 BASIC.

Following a module RESET, the memory is tested, cleared where necessary, and internal system variables subsequently initialized.
BASIC assigns the highest possible CP RAM address ($7FFF_H$) to the MTOP system variable and uses this number as base value for the random generator. If there is no backup battery, defaults for the date and time are entered in IW0 to IW3 of the module's process input image (PII).

Note:

Note that the module defaults to (V.24) after a RESET when no interface parameters are available. This means that an input device must be connected to the CP 521 BASIC over the (V.24) interface.

Note:

When the device is switched on for the first time, the message ”Invalid line number” may appear. This can happen when operating the module without a backup battery.
Remedy:
Use a backup battery and switch the device off and on again.

The CP 521 BASIC already contains a BASIC program/is already initialized
If the CP 521 BASIC already contains a BASIC program, its restart characteristics are those listed in Table 5-1. ”PROG” commands can be used to store the interface parameters for a module restart and, in conjunction with the mode selector switch, to initiate an automatic program start.

Table 5-1. Restart Characteristics of the CP 521 BASIC with BASIC Program/ Following Initialization

Position of the CP 521 BASIC's Mode Selector	Last "PROG" Command Issued		
	PROG	PROG 1	PROG 2
PROG	Command Mode	Command Mode	Command Mode
RUN	Command Mode	Start the BASIC program stored in RAM, if available	Start the BASIC program with the No. 1 stored on the memory submodule, if available

Interface parameters
- The PROG command sets the interface parameters to their default values.
  The Auto Baud routine (”space” key) automatically identifies the baud rate for the bidirectional interface.
- PROG1/PROG2 initializes the bidirectional and unidirectional interfaces with the parameters that were active when PROG1 or PROG2 was last issued.

If no program is available, the module enters the Command mode. In this mode, you can enter commands, write a program, or start a program (5.3.1 and 5.3.2).

Automatic restart of the CP 521 BASIC using the AUTOINIT function

It is possible to perform a restart without battery backup provided the following conditions are met:

- A memory submodule containing at least one BASIC program is plugged into the CP 521 BASIC.
- The string ‘AUTOINIT’ is entered in the comment line of any BASIC program on the memory submodule.

- **Entering the ‘AUTOINIT’ string:**
  Enter the ‘AUTOINIT’ string immediately after the REM command:
  - A maximum of one space is permitted between REM and AUTOINIT.
  - Either uppercase or lowercase can be used.

  Examples:

  ```
  >11 REM AUTOINIT ...
  ```

  or

  ```
  >11 REMAUTOINIT ...
  ```

- **Automatic restart ‘AUTOINIT’ without battery backup:**
  If both conditions are met, the restart characteristics of the CP 521 BASIC are as follows:
  1. The Auto baud routine (wait for input of a space via the terminal) is skipped.
  2. The memory (RAM) of the CP 521 BASIC is deleted.
  3. The CP 521 BASIC initializes itself with the default parameters.
  4. The CP 521 BASIC goes to command mode regardless of the position of the PROG/RUN switch.
  5. No string (prompt) is output to the serial interface.
  6. You can now start any BASIC program on the memory submodule using the CPU request $D0_H$.

- **Automatic restart ‘AUTOINIT’ with battery backup:**
  If the ‘AUTOINIT’ string is found, the RAM of the CP 521 BASIC is deleted, regardless of whether the CP 521 BASIC has a backup battery or not. Any BASIC program stored in RAM of the CP 521 BASIC before power failure will be deleted.
  The restart is otherwise the same as that described under ‘Automatic restart ‘AUTOINIT’ without battery backup’.

- **Simultaneous use of ‘AUTOINIT’ and PROG1/PROG2**
  If you have being working in backup mode until now and have used the PROG1 or PROG2 function, the PROG1 or PROG2 function will be processed as before - even if the BASIC program is supplemented with ‘AUTOINIT’ - provided the battery backup is functioning properly.
  If the BASIC program is lost because of a lack of battery backup, it will also not be possible to process PROG1 or PROG2. The AUTOINIT function is then used (as described above). The RAM of the CP 521 BASIC is deleted and PROG1/PROG2 are ignored.

5.3 Programming

The module can operate in two modes, i.e. direct (Command) mode and interpreter (RUN) mode. Commands can be entered only when the CP 521 BASIC is in Command mode. When a command is entered, the interpreter executes it immediately. In all subsequent portions of this manual, the two CP 521 BASIC modes are referred to as RUN mode and Command mode.

5.3.1 Command Mode

In Command mode, commands are executed immediately and the results displayed. BASIC programs are also written in this mode.
For the Command mode to be activated, either
- the mode selector on the CP 521 BASIC must be set to ”PROG” during the runup or
- the program currently executing must be aborted with <CTRL C>.

The Command mode can also be used to edit existing programs, store programs on an EEPROM or RAM submodule, or start existing programs.
The COM software is required in order to store programs on EPROM (5.4).

Programs can be transferred from the programmer to the CP 521 BASIC, or vice versa, only in Command mode.

5.3.2 RUN Mode

You can initiate execution of your programs in RUN mode. To activate the RUN mode, either
- the RUN command must be entered in the Command mode
- the mode selector must be at ”RUN” during the module runup and a program must be available on the memory submodule (5.7.3), or
- CPU request $D0_H$ must be forwarded to the CP 521 BASIC.

You can abort any program by

- entering <CTRL C> on the terminal (unless measures were taken in the program to preclude this!).

Program errors are flagged on the input terminal and on the CPU interface.

5.4 COM Software

The COM software for the CP 521 BASIC provides support for programming the CP 521 BASIC module and for storing your programs on a memory submodule, hard disk or floppy disk. The COM software can be invoked on a programmer under S5-DOS.

The first step in familiarizing you with the use of the COM software is to explain how to transfer the file from the floppy to your programmer's hard disk and start the software.

Transferring the COM software to the programmer's hard disk

- MS/DOS operating system
  Load operating system
  Insert COM diskette in drive A of the programmer
  Call up file "INSTALL.EXE" on the COM diskette
  You will be guided by the menu afterwards.

- PCP/M operating system
  Load operating system
  Insert COM diskette in drive A of the programmer
  Copy the file "S5PEC21X.CMD" into the user area 0 of the hard disk (PCOPY a:\COM_521B\S5PEC21X.CMD).

  You can assign the following attributes to the file:
  - [RO] to protect it against unintentional overwriting
  - [SYS] to enable the file to be invoked from any user area.

Starting the COM software

Invoke the S5 command interpreter (enter S5 and press <CR>).
Set the cursor to ”COM521 BASIC” in the ”Package Select” menu.
Select this package with <F1>.

Note:

The COM software ”S5PEC21X.CMD” must be on the hard disk.

The following screen form is displayed:

COM software start form on the programmer or PC

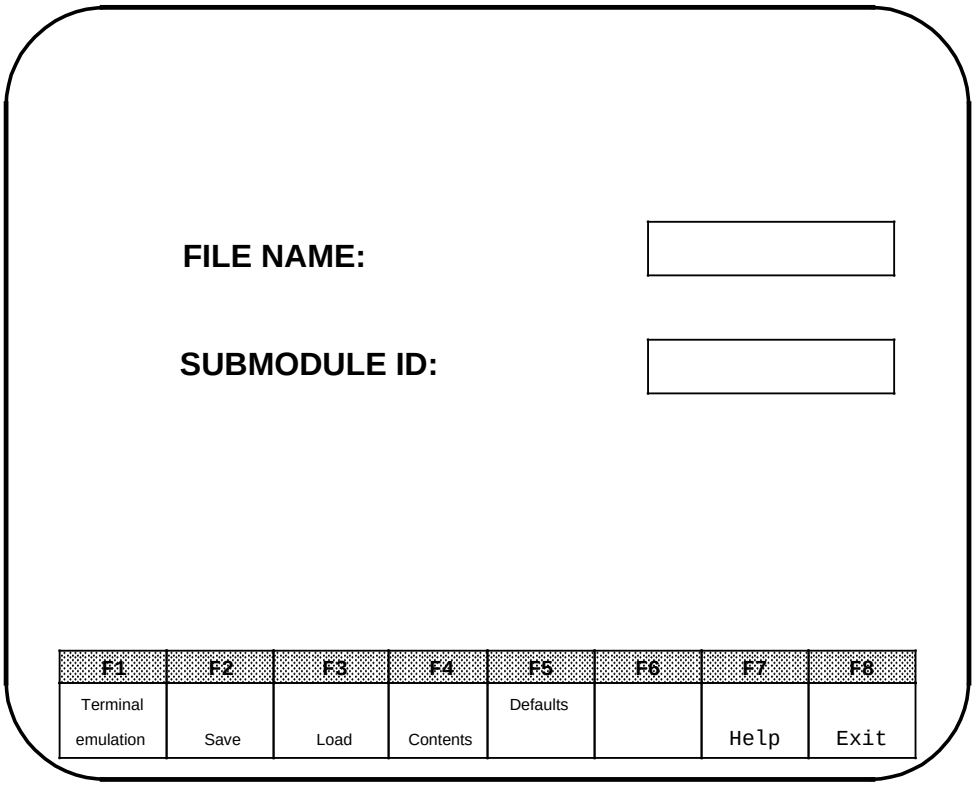


Figure 5-1. The COM Software's Start Menu

F1: Enable terminal emulation
F2: Branch to the SAVE menu
F3: Branch to the LOAD menu
F4: Branch to the CONTENTS menu
F5: Enter filename or submodule identifier
F7: Help menu for the current menu
F8: Terminate program

You can make the following selections in the start form:

- Terminal emulation
- Save and load existing programs
- Read the contents of text files
- Display the contents of the hard disk, the floppy and the memory submodule

Before continuing with COM, you should enter the default values for the program. Default values are required for storing and reaccessing user programs. Programs are stored on the floppy or hard disk under the specified filename (”FILENAME”). ”SUBMODULE ID” identifies the submodule on which the programs can be stored. The ”Defaults” menu can be invoked in any other menu by pressing function key <F5>.

DEFAULTS menu

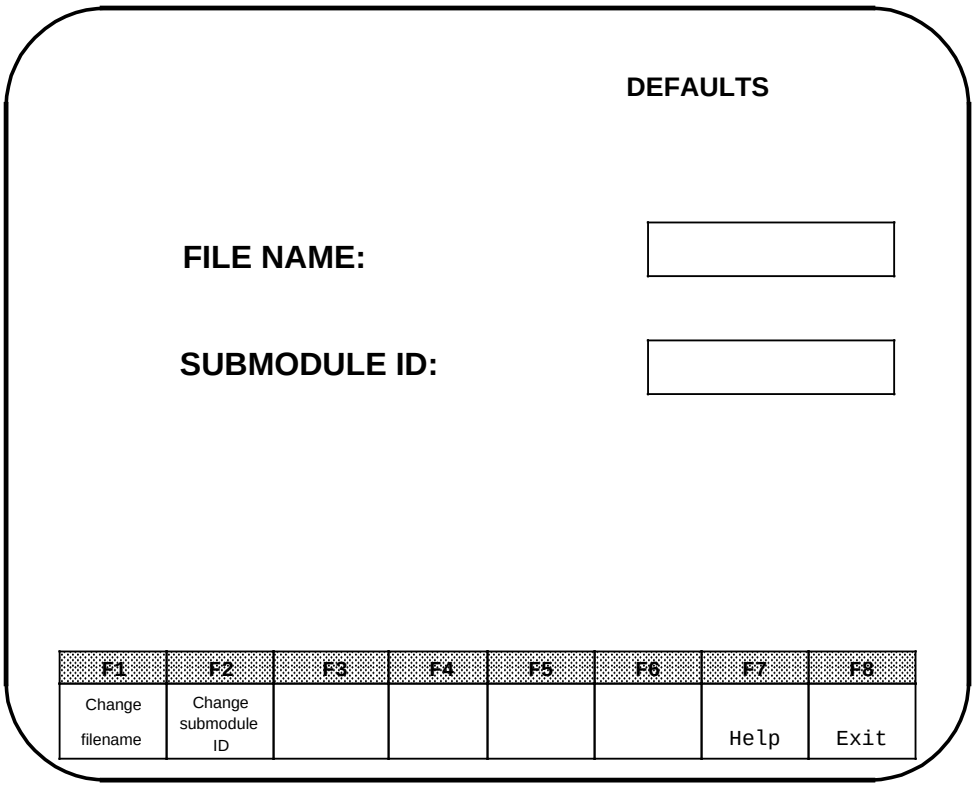


Figure 5-2. The COM ”Defaults” Menu

F1: Change filename
F2: Change submodule ID
F7: Help menu for the current menu
F8: Return to preceding menu
After you have entered a filename or submodule identifier, the default values are displayed at the upper left in all other menus with the exception of the Terminal Emulation menu.

5.4.1 Terminal Emulation

When you select ”Terminal emulation” (<F1>), the programmer assumes the same performance characteristics (that is, emulates) a terminal, i.e. it forwards keyboard entries to the CP 521 BASIC and echoes them on the screen. To initiate error-free communication between CP 521 BASIC and programmer, press function key <F1> in the Terminal Emulation menu. The CP 521 BASIC then ”tunes” itself to the programmer (protocol).
Terminal emulation thus means that commands can be forwarded to the CP 521 BASIC and executed immediately, or they can be entered as program when the CP 521 BASIC is in the Command mode.
Unless the program contains interlocks precluding the use of <CTRL C>, this key combination can be used to abort and then edit any program.
To exit the terminal emulation menu and return to the top menu, press function key <F8> on the programmer.

TERMINAL EMULATION menu

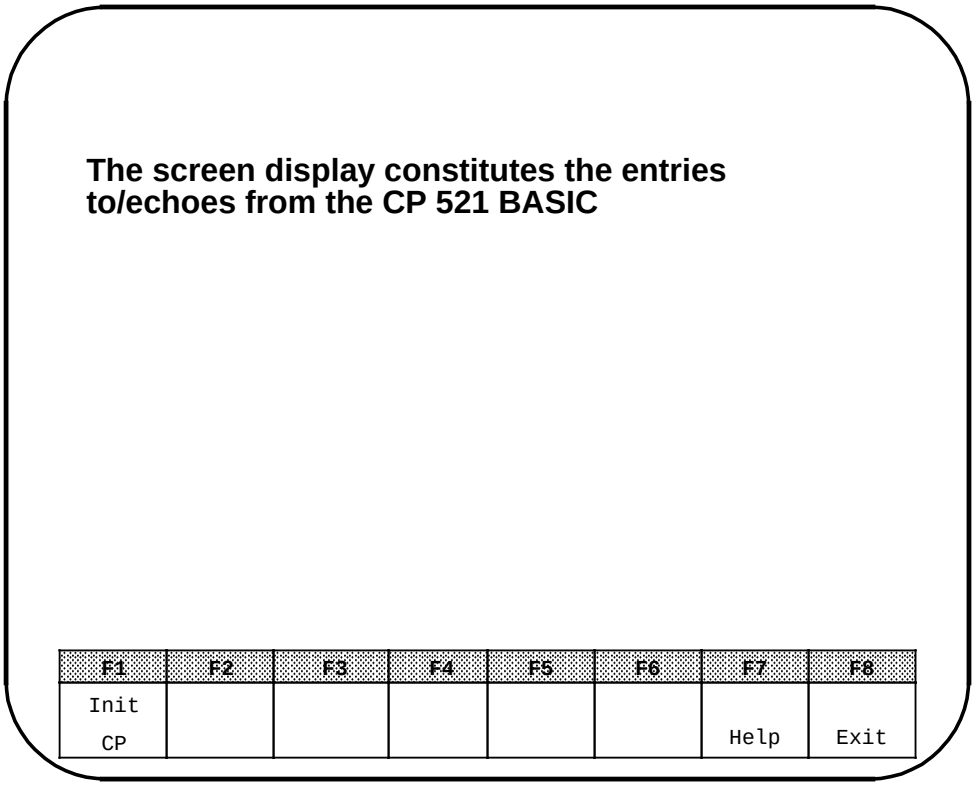


Figure 5-3. The COM ”Terminal Emulation” Menu

F1: CP initialization
F7: Help menu for the current menu
F8: Return to the top menu

5.4.2 Save BASIC Programs

The ”Save” menu allows you to save BASIC programs written on the module in Command mode to hard disk or floppy.
The programs are transferred to the storage medium in different forms, depending on which function key is pressed in the ”Save” menu:

- ”Source code PG”: Readable form
- ”Obj. code PG”: Compressed
- ”CP contents PG”: Dump contents of memory submodule to the programmer

SAVE menu

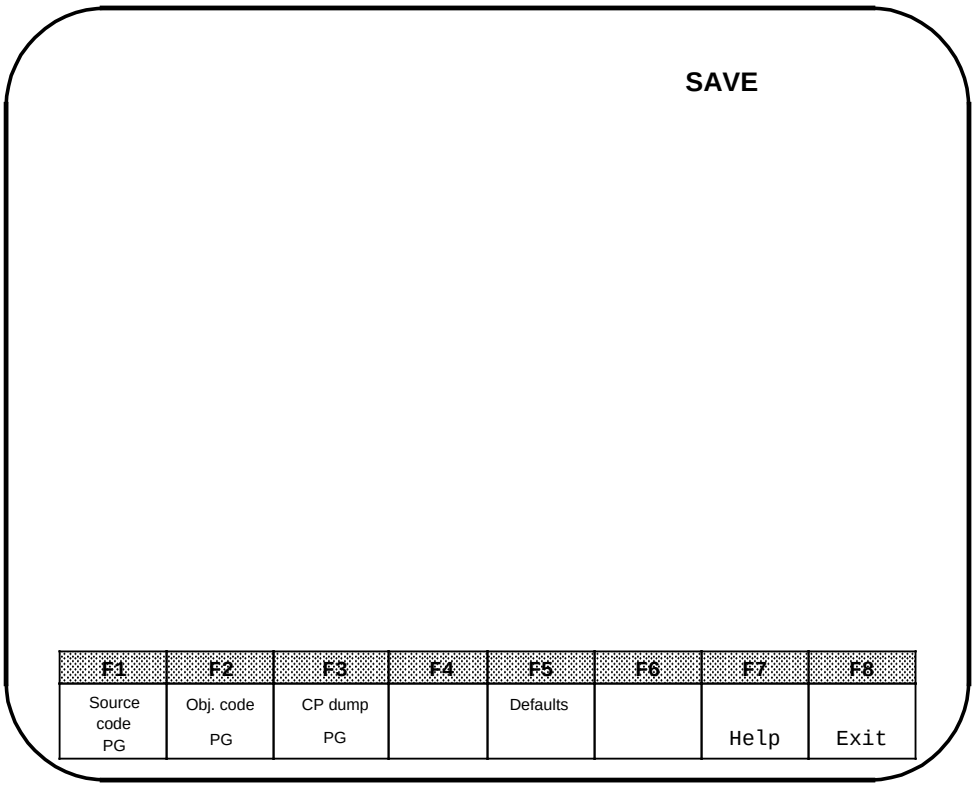


Figure 5-4. The COM ”Save” Menu

F1: Transfer the current BASIC program from the CP 521 BASIC to the programmer's/PC's mass storage in ASCII code
F2: Transfer the current BASIC program from the CP 521 BASIC to the programmer's/PC's mass storage in compressed form
F3: Dump the entire contents of the CP 521 BASIC's memory submodule to the programmer's/PC's mass storage.
F5: Enter filename or submodule identifier
F7: Help menu for the current menu
F8: Return to the top menu

”Source code PG” (<F1>)

When you press function key <F1> (”Source code PG”), the CP 521 BASIC transfers the BASIC program last activated to the programmer in ASCII code. The program is stored under the specified filename with the extension ”.BAS”, and can now be edited using a text editor.
If no BASIC program is available, an error message is displayed on the programmer's monitor.

”Obj. code PG” (<F2>)

If you press function key <F2> (”Obj. code PG”), the CP 521 BASIC compresses the BASIC program last activated before transferring it to the programmer's or PC's mass storage.
The program is stored under the specified file name with the extension ”.TOK”, and can now be used to program EPROMs. If no BASIC program is available in the CP 521 BASIC, an error message is displayed on the programmer's monitor.

”CP contents PG” (<F3>)

If you press function key <F3> (”CP contents PG”), the module dumps the entire memory submodule to the programmer. The dump is stored under the specified file name with the extension ”.ALL”.
An error message is displayed if no memory submodule is plugged into the CP 521 BASIC.

5.4.3 Loading BASIC Programs

Both RAM-resident and memory submodule-resident BASIC programs can execute on the CP 521 BASIC.
Two options are available for transferring BASIC programs to the CP 521 BASIC:

- Transfer programs from the programmer to the CP 521 BASIC
- Program a memory submodule on the programmer and plug it into the CP 521 BASIC

The menu for loading BASIC programs is selected by pressing function key <F3> (”Load”) in the top menu.

LOAD menu

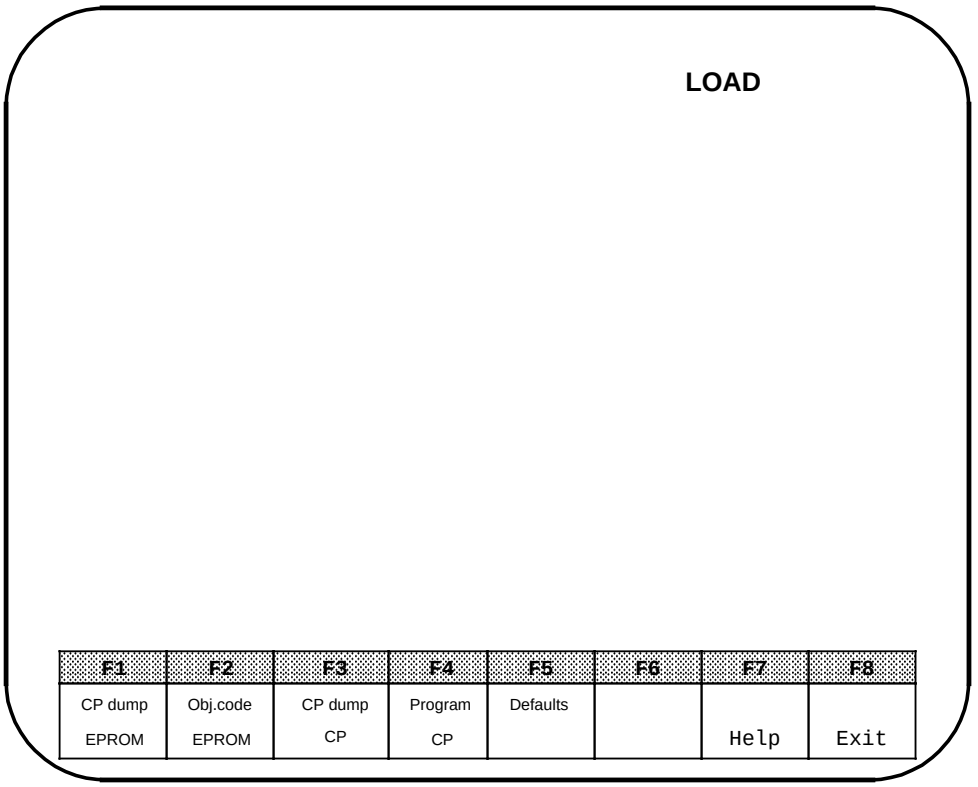


Figure 5-5. The COM ”Load” Menu

F1: Programming of a memory submodule with a file from the programmer's mass storage which contains a dump of the CP 521 BASIC's memory submodule (.ALL).

F2: Programming of a memory submodule with a file from the programmer's mass storage which contains an executable program (.TOK).

F3: Downloading of a complete dump from the programmer's mass storage to the CP 521 BASIC (.ALL)

F4: Downloading of a BASIC program from the programmer's mass storage to the CP 521 BASIC (priority: .TOK if available, otherwise .BAS)

F5: Enter filename or submodule identifier

F7: Help menu for the current menu

F8: Return to the top menu

Transfer program from the programmer to the CP 521 BASIC
”Program CP” (<F4>)

When you press <F4> (”Program CP”), the program with the specified filename is transferred to the CP 521 BASIC.
The file extension (”.BAS” or ”.TOK”) tells the COM software whether the program is in ASCII code or compressed form. If both are available, the compressed program takes priority. If an ASCII program is transferred, it is stored in compressed form on the CP 521 BASIC to save space.

”CP dump CP” (<F3>)
When you press <F3> (”CP dump CP”), the file with the extension ”.ALL” is loaded into the module. Files with an ”.ALL” extension always contain a complete dump, and require that the CP 521 BASIC be equipped with an appropriate memory submodule.

Programming memory submodules
”Obj. code EPROM” (<F2>)

When the ”Obj. code EPROM” function is selected, the program stored under the specified filename is written to an EPROM. The first time a blank EPROM is programmed, the programmer tags the submodule as BASIC submodule for the CP 521 BASIC.
If any other programmed submodule is plugged in, an error message is displayed.
The memory submodule was selected in the Defaults menu by the entry of a submodule ID (see Help menu).
The BASIC program must be available on the programmer or PC as compressed file with ”.TOK” extension.
When the EPROM has been programmed, the program number assigned to the program is displayed on the monitor. The memory submodule can now be plugged into the CP 521 BASIC and invoked with the ROM or RROM [integer] command (5.7.1). The [integer] parameter corresponds to the program number under which the program was stored.

”CP dump EPROM” (<F1>)
This function programs the EPROM with a previous CP dump. Memory submodule and source file must be specified in the ”Defaults” menu. The dump must be located on the file with the ”.ALL” extension.

5.4.4 Displaying the Contents of Memory and Files

Pressing <F4> in the top menu screens the ”Contents” menu. The function keys for this menu allow you to display the contents of various storage media or files.

”Drive” (<F1>)
Press <F1> to display the directory of the specified drive. The display includes only files with a ”.TOK”, ”.BAS” or ”.ALL” extension.

”EPROM” (<F2>)
Press <F2> to display the numbers of the programs on the EPROM submodule.

”File” (<F3>)
Press <F3> to display the contents of the workfile with the extension ”.BAS”, thus allowing you, for instance, to locate a specific BASIC program. This file must be edited using a text editor; it cannot be edited with COM.

CONTENTS menu

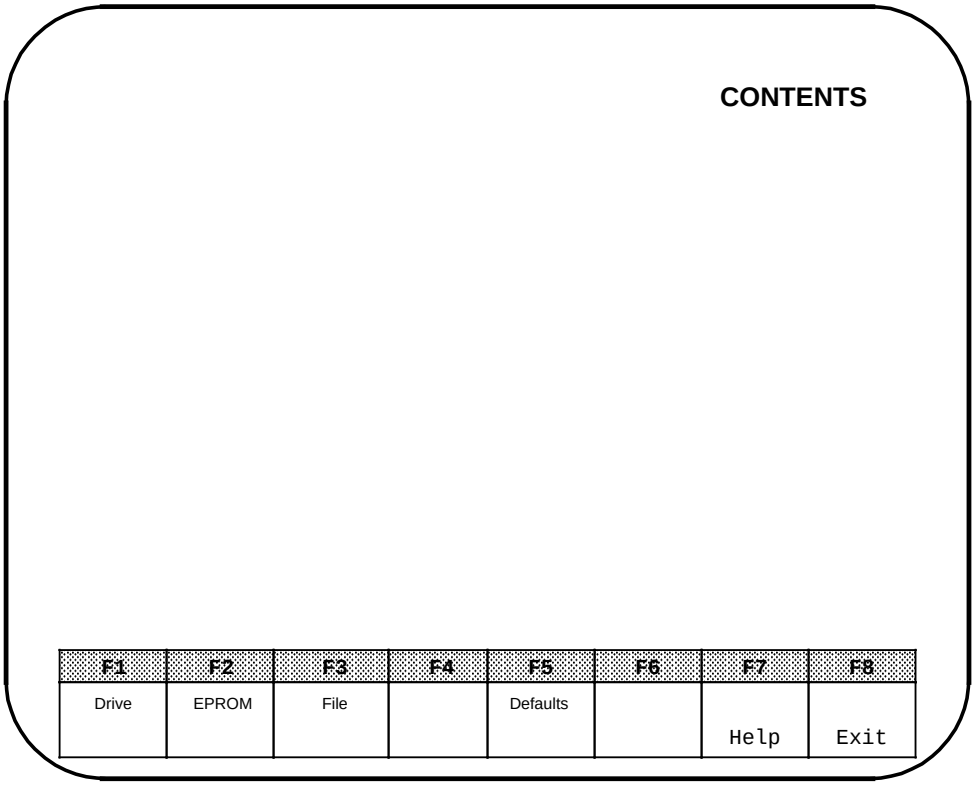


Figure 5-6. The COM ”Contents” Menu

F1: Display the drive directory
F2: Display the numbers of the programs on the EPROM
F3: Display file contents
F5: Enter filename or submodule ID
F7: Help menu for the current menu
F8: Return to the top menu

5.5 The BASIC Interpreter

The objective of the next several sections is to provide introductory material on the BASIC interpreter, which is based on the BASIC interpreter from Intel implemented in the 8050A$_H$'s ROM.
The various sections deal with the different data types, instructions and statements; in short, with everything required for programming. They do not, however, constitute an introduction to the BASIC programming language.
The BASIC version covered in these sections incorporates most of the ”standard BASIC dialects” while also including a number of operations for communication with the CPU and communication over the serial interfaces.

5.5.1 Statements/Format Statements

Statements
In BASIC, a program consists of program lines. Each program line begins with a line number. The line number is followed by the actual statement; the line ends with a carriage return (CR) character. In lines containing more than one statement, the statements are separated from one another by a colon (:). Some statements can be executed in Command mode. In this manual, each statement description indicates whether that statement can be execute in Command mode or only in RUN mode. In BASIC, there are three fundamental types of statements: Assignment statements, input/output statements and control statements. The description of a specific statement indicates the type to which that statement belongs.

- Each line in a program must begin with a line number between 0 and 65535.
- BASIC uses the line numbers to arrange the program statements in the proper order.
- A specific line number may occur only once within a given program.
- The statements need not be entered in order, as BASIC automatically sorts them into ascending sequence.
- A program line may contain no more than 79 characters (including the line number).
- Blanks (spaces) are ignored or added automatically in a LIST.

- A line may contain only ***one*** line number, but several statements; the statements must be separated from one another by a colon (:).

Format statements
Format statements may be used only with a PRINT statement. Format statements are TAB([expr]), SPC([expr]), USING [special char.] and CR (carriage return without line feed). The format statements are discussed in detail in the description of the PRINT statement.

5.5.2 Data Formats

Numbers can be represented in two different ways in BASIC:
- Floating-point representation
- Integer representation

Floating-point representation
The following number range can be processed in BASIC:
±1.0E - 127 to ±0.99999999E +127.

BASIC works with eight significant digits. Internally, all numbers are rounded to accord with this degree of precision. Numbers can be input and output in four formats: integer format, decimal format, hexadecimal format and exponential format. Example: 127, 34.98, 0A6EH, 1.23456E+3.

Constants
Constants are floating-point numbers in the range:
±1.0E - 127 to ±0.99999999E +127.
In this manual, constants are identified as follows:

```
[const]
```

Integer representation
In BASIC, integers are all numbers from 0 to 65535 or $0FFFF_H$. Integers can be entered in decimal or hexadecimal format, whereby all hexadecimal numbers must begin with a valid digit (the number $A000_H$, for instance, must be entered as $0A000_H$). When operators (for instance .AND) are used which require integers, BASIC truncates the decimal portion of the number to obtain an integer.

Line numbers are always integers. In this manual, integers and line numbers are identified as follows:

`[integer]` or `[ln num]`

Note:
In this manual, brackets [ ] are used only to identify an integer, a constant or the like. Brackets must **never** be used when entering a number or variable. Options are identified by { }.

5.5.3 Operators

An operator executes a specific operation on variables and/or constants. Operators require one or two operands. Typical ”dyadic” operators, i.e. operators requiring two operands, are addition (+), subtraction (-), multiplication (*) and division (/). Operators requiring only one operand are often referred to as unary operators. Typical unary operators are SIN, COS and ABS.

5.5.4 Variables

Variable names must always begin with a letter. The remaining characters may be letters, digits and/or the underline character. These combinatorial options make it possible to define expressive names for specific variables. Some permissible variable names in BASIC are:

```
FRED        AMPERE1        I_I1       ARRAY(ELE_1)
```

Note:
Each character in a variable reserves one byte. You should therefore keep these names as short as possible.

Note:
Please note that BASIC uses only the first character in the name, the last character in the name, and the length of the name for internal referencing of a variable. The interpreter is therefore not able to distinguish between ”LAENGE0” and ”LIEFER0”.

Note the following carefully as regards variable names:

- The longer a variable name, the longer BASIC needs to process that name.
- Reserved words must never be used as part of a variable name (the variable TABLE, for example, may not be used because TAB is a reserved word).

The system reports ERROR: BAD SYNTAX when the user attempts to define a variable containing a reserved word.
The system reports ERROR: BAD SYNTAX when the user attempts to define a variable containing a reserved word.
Variables containing a one-dimensional expression [expr] as subscript (e.g. A(0)) are often referred to as dimensioned variables or array variables. Variables comprising only a variable name or subscript are referred to as scalar variables. Dimensioned variables are discussed in detail in the description of the DIM statement routine (5.8.7). In this manual, variables are identified as follows:

```
[var]
```

In BASIC, variables are assigned ”statically”. The first time a variable is used, BASIC reserves a memory area (8 bytes) specifically for that variable. This memory area cannot be used for the variable in all instances. Following a statement such as Q=3, for example, it is not possible to indicate that variable Q is no longer needed and that BASIC should therefore release the eight bytes assigned to it. The only way to release a memory area assigned to a variable is to execute a CLEAR statement (5.8.1). This statement releases the entire memory area set aside for variables, i.e. all variables are deleted.

Note:
As compared to a dimensioned variable, BASIC requires considerably less time to locate a scalar variable. It is not necessary to first compute an expression in a scalar variable. For this reason, dimensioned variables should be used only when absolutely necessary if a program is to execute quickly. Scalar variables can be used for intermediate results and the final result then assigned to a dimensioned variable.

5.5.5 Expressions/Relational Expressions

Expressions
An expression is a logical mathematical formula containing operators (unary and dyadic), constants and variables. Expressions may be simple or extremely complex, for instance 12*EXP(A)/100, H(1)+55 or (SIN(A)*SIN(A)+COS(A)*COS(A))/2.
A ”single” variable [var] or constant [const] is also regarded as an expression. In this manual, expressions are identified as follows:

```
[expr]
```

Relational expressions
Relational expressions contain the operators EQUAL TO (=), NOT EQUAL TO (<>), GREATER THAN (>), LESS THAN (<), GREATER THAN OR EQUAL TO (>=) and LESS THAN OR EQUAL TO (<=). They are used in control statements to ”test” a condition (e.g. IF A < 100 THEN ...). Relational expressions always require two operands. In this manual, relational expressions are identified as follows:

```
[rel expr]
```

5.5.6 System Control Parameters

The system control parameters include:
- LEN (length of the program),
- FREE (the maximum number of bytes available in RAM for variables and strings) and
- MTOP (the last memory location reserved for BASIC).

5.5.7 Stacks

BASIC reserves 512 bytes in data memory for two ”software” stacks. These two stacks are referred to as the control stack and the argument stack. Users who want only to program in BASIC do not need to know details on how these stacks work.

Control stack
This stack is used to buffer all information pertaining to loop control (such as DO-WHILE, DO-UNTIL and FOR-NEXT) and to BASIC subroutines (GOSUB). The nesting depth is nine for FOR-NEXT and 52 for all other operations.

Argument stack
This stack is used to buffer all arguments, values and intermediate results with which BASIC is currently working. Operations such as addition, subtraction, multiplication and division are executed with the aid of the argument stack.

5.5.8 Line Editor

BASIC contains a minimum-scope line editor. If it is necessary to edit a line that has already been input, that line must be reentered in its entirety. It is possible, however, to delete individual characters as long as the line has not yet been completed. To do so, enter ”DEL” ($7F_H$). The ”DEL” ($7F_H$) character clears the last character entered in the text input buffer. CTRL-D deletes the entire line. BASIC recognizes the CTRL-Q (X-ON) and CTRL-S (X-OFF) characters. When entering data, care must be taken that CTRL-S is not entered unintentionally, as BASIC would then no longer respond to input from the console. To counteract CTRL-S, enter CTRL-Q.

Note:
In this manual, a carriage return is identified by the symbol (CR). On most keyboards, the carriage return character is invoked by pressing RETURN.

5.6 Command Descriptions

The following commands can be entered in Command mode only.

5.6.1 RUN

Command	RUN(cr)

Description
RUN(cr) sets all variables to zero, clears the interrupts and the buffers to the S5-CPU and the I/Os, and restarts the selected program at the first line number. The BASIC interpreter can be forced out of the Command mode only with RUN or into the RUN mode with a GOTO statement. The program run can be aborted at any time by entering CTRL-C on the console.

Special features
In contrast to other BASIC interpreters, which permit entry of a line number in the RUN command (e.g. RUN 100), BASIC does not allow a line number entry in the RUN command. The program run always begins with the first line number. In order to ”simulate” a RUN [ln num], the GOTO statement [ln num] can be used entered in Command mode. For details, refer to the description of the GOTO statement (5.8.13).

Example

```
>10 FOR I=1 TO 3
>20 PRINT I
>30 NEXT I
>RUN

 1
 2
 3

READY
>
```

5.6.2 CONT

Command	CONT(cr)

Description
Programs that can be aborted by entering CTRL-C on the console or executing a STOP statement (5.8.29) can be resumed by entering CONT(cr). Variables can be displayed or modified after aborting and before resuming program execution. It is not possible, however, to resume program execution with CONT when the program was aborted due to an error or when it was modified in the interim, nor can the program be resumed with CONT following entry of a RAM or ROM command.

Special features
None

Example

```
>10 FOR I=1 TO 10000
>20 PRINT I
>30 NEXT I
>RUN

 1
 2
 3
 4
 5    - (TYPE CONTROL-C ON CONSOLE)

STOP - IN LINE 20

READY
>PRINT I
 6

>I=10

>CONT

 10
 11
 12
```

5.6.3 LIST

Command	`LIST {ln num{-ln num}} (cr)`

Description
The LIST(cr) command outputs the program to the console. Note that the LIST command ”formats” the program for easy readability. Blanks are inserted behind the line number and in front of and behind the statements. This feature supports troubleshooting in BASIC programs. LIST can be aborted at any time by entering CTRL-C on the console.

Special features
There are two versions of the LIST command:

`LIST[ln num] (cr)` and
`LIST[ln num] - [ln num] (cr)`

The first lists the program from the specified line number (integer) to the end, the second from the first line number (integer) to the second line number (integer).

Note:
The two line numbers in the second option must be separated from one another by a hyphen.

Example

```
READY
>LIST
 10 PRINT ″LOOP PROGRAM″
 20 FOR I=1 TO 3
 30 PRINT I
 40 NEXT I
 50 END

READY
>LIST 30
 30 PRINT I
 40 NEXT I
 50 END

READY
>LIST 20-40
 20 FOR I=1 TO 3
 30 PRINT I
 40 NEXT I
```

5.6.4 LIST#

Command	`List# {ln num{- ln num}} (cr)`

Description
The LIST#(cr) command outputs the program over the unidirectional interface. All parameters for LIST also apply for LIST#.

5.6.5 NEW

Command	`NEW (cr)`

Description
NEW(cr) deletes the program currently in RAM, sets all variables to zero, resets all interrupts, and clears the buffers to the S5-CPU and the I/Os. String variables are reset, but the memory area reserved for them with the ”String” command is not released. The timer is not affected.

5.7 Memory Commands

The CP 521 BASIC can store several programs on a memory submodule. The number of programs which can be stored on the submodule is limited only by its storage capacity. The programs are stored on the memory submodule in ascending order, and can be invoked and executed when required. The storing of programs in ascending order is referred to as submodule file. The following commands are provided for selecting, generating and processing the submodule file.

5.7.1 RAM and ROM

Commands	`RAM(cr) und ROM [integer](cr)`

Description
These two commands tell the BASIC interpreter whether the current program (the current program is the program output with LIST or invoked by RUN) is located in RAM or on the memory submodule.

Commands	`RAM(cr)`

When RAM(cr) is entered, the interpreter selects the current program from RAM. This is usually regarded as the ”standard” mode, and the majority of users invoke the command interpreter in this mode.

Command	`ROM [integer](cr)`

If ROM[integer](cr) is entered, BASIC selects the current program from the memory submodule. If the command is entered without an integer (i.e. ROM(cr)), BASIC defaults to ROM 1. Since the programs are stored on the memory submodule in ascending order, the integer entered as parameter in the ROM command specifies the program you want to invoke or list. ERROR: PROGRAM NOT FOUND is reported if an attempt is made to select a non-existent program (example: ROM 8 was entered although only six programs are stored on the submodule).

The ROM command does not transfer programs from the memory submodule to RAM. It is therefore not possible to EDIT programs in ROM mode. ERROR: CAN'T EDIT ON MEMORY MODULE is reported if an attempt is made to edit a program by entering a line number while in ROM mode. The XFER command, which is described in Section 5.7.2, transfers programs from the memory submodule to RAM for editing.

Since the ROM command does not transfer programs to RAM, the memory submodule and RAM may contain different programs at one and the same time. The user can switch back and forth between RAM and ROM mode as required. The fact that programs need not be transferred to RAM provides yet another advantage, i.e. that the entire RAM can be used for storing variables as long as the program itself is on the memory submodule. The system control parameters MTOP and FREE always refer to RAM, never to the memory submodule.

Special features
None

5.7.2 XFER

Command	`XFER(cr)`

Description
The XFER (Transfer) command transfers the current program from the memory submodule to RAM and invokes the RAM mode. If XFER is entered while BASIC is in RAM mode, the program stored in RAM is transferred to RAM and the RAM mode invoked. Following execution of the XFER command, the program in RAM can be edited just like any other RAM program.

Special features
None

5.7.3 PROG

Command	`PROG(cr)`

Description
PROG transfers the current program to the memory submodule. The program must be located in RAM when the PROG command is issued.
When PROG(cr) is entered, BASIC displays the number of the submodule file under which the program is stored.

Note:

The CP 521 BASIC does not support programming or erasing of EPROMs. Programming and/or erasing of EPROMs requires a COM software package and a programmer.

”CAN'T PROGRAM AN EPROM MODULE” or ”UNKNOWN MODULE/NO BASIC MODULE” is displayed if an attempt is made to program an EPROM with one of the commands described in the following or if the CP 521 BASIC was not equipped with a submodule.
If an unknown memory submodule is to be used to store programs, the CP 521 BASIC prompts you to indicate whether or not the memory submodule is to be overwritten.

Example

```
>LIST
 10 FOR I=1 TO 10
 20 PRINT I
 30 NEXT I

READY
>PROG
 12

READY
>ROM 12

READY
>LIST
 10 FOR I=1 TO 10
 20 PRINT I
 30 NEXT I

READY
>
```

According to the example above, the program just stored is the twelfth program on the memory submodule.

Special features
None

5.7.4 PROG1 and PROG2

Command	`PROG1(cr) und PROG2(cr)`

Description

Command	`PROG1(cr)`

Following power-up, you must normally initialize the module's bidirectional interface by pressing the space bar. For the purpose of simplification, BASIC has a command called PROG1 which enters the baud rate info into the battery-backed RAM. The next time the module is ”powered up”, i.e. on a RESET, the CP 521 BASIC uses this information to initialize the bidirectional interface. The program in RAM is then started.

Command	`PROG2(cr)`

The PROG2 command is the same as the PROG1 command except that it invokes the program with the number 1 from the memory submodule immediately following a RESET.

Note:

A program can be started with PROG1 or PROG2 only when the CP 521 BASIC's mode selector is at RUN; if PROG2 is used to start a program, the CP 521 BASIC must also be equipped with a memory submodule.
UNKNOWN MODULE/NO BASIC MODULE is displayed if the CP 521 BASIC is equipped with no memory submodule or with an invalid memory submodule.
If the memory submodule is an EPROM, the interpreter flags an error when PROG1 or PROG2 is entered, but the information needed for a restart is still stored.

When PROG1 or PROG2 is entered, BASIC displays the number of the submodule file under which the program is stored.
PROG1/PROG2 make it possible to invoke a program with RUN immediately following a RESET without the module having to be interfaced to a terminal. PROG2 corresponds to a ROM 1, RUN sequence. This is ideal for control applications for which a terminal is not always readily available. This feature allows you to generate a special initialization sequence in BASIC.
When PROG (without parameters) is used, the CP 521 BASIC executes the Auto Baud routine (confirmation of the space bar) on the next power-up (RESET).

5.7.5 ERASE

Command	`ERASE [integer]`

Description
The program with the ”program number” entered in the ERASE command is tagged as having been erased. An error message is displayed if an invalid program number is specified (<1 or >254).

Special features
None

5.7.6 REORG

Command	`REORG (cr)`

Description
The REORG command physically removes the programs that were tagged as having been erased. The REORG command compresses memory, and the remaining programs thus follow one another contiguously and without gaps. The remaining programs retain their program numbers. The numbers of programs that have been physically erased are released.

Special features
None

Note:

The CP must not be switched off during a REORG, as this could destroy the programs on the memory submodule.

5.8 Statement Descriptions

The statement descriptions are presented in three-line tabular form.

- Line 1: Describes the statement syntax.
- Line 2: Specifies the mode in which the statement can be used.
- Line 3: Classifies the statement, i.e. indicates whether it can be used for program control, for input/output functions or for other purposes.

5.8.1 CLEAR

Statement	CLEAR
Mode	COMMAND and/or RUN
Type	Control

Description

The CLEAR statement sets all variables to zero, clears the internal buffers, and resets the interrupts and the stacks. This means that an ONTIME statement (5.8.19) must be executed after CLEAR before BASIC will once again accept interrupts. By the same token, software errors cannot be intercepted via an ONERR statement (5.8.18) until an ONERR [integer] statement has executed. The CLEAR statement does not affect the timer started by the CLOCK1 statement (5.8.20), nor does it reset the memory area reserved for strings, thus making it unnecessary to enter the STRING [expr],[expr] statement (5.12.1) to reserve space for strings after executing a CLEAR. Normally, CLEAR is used simply to ”reset” all variables.

Special features

None

5.8.2 CLEARI and CLEARS

Statements	CLEARI and CLEARS
Mode	COMMAND and/or RUN
Type	Control

Description

Statement	CLEARI

All interrupts activated by BASIC are reset (ONTIME). This does not affect the CP 521 BASIC's timer. You can use CLEARI to disable the ONTIME interrupt during execution of specific program sections and reenable it afterward with the ONTIME command.

Statement	CLEARS

CLEARS resets the interpreter's stacks (argument stack and control stack). It is sometimes necessary to reset the stacks when an error occurs in a subroutine (GOSUB) or in a program loop (DO-WHILE, FOR-NEXT, DO-UNTIL) in order to enable the program to continue ”normally”. CLEARS can also be used to exit program loops prematurely (DO-WHILE, FOR-NEXT, DO-UNTIL). Arguments pushed onto the argument stack with PUSH are lost.

Example

```
>5 REM TEST OF CLEARS AND CLEARI
10 PRINT ”MULTIPLICATION TEST, YOU HAVE 5 SECONDS”
>20 FOR I = 2 TO 9
>30 N = INT(RND*10) : A = N*I
>40 PRINT ”WHAT IS UP”, N, ”*”, I, ”?”: CLOCK1
>50 TIME = 0 : ONTIME 5, 200 : INPUT R: IF R<>A THEN 100
>60 PRINT ”THAT IS CORRECT”: TIME=0: NEXT I
>70 PRINT ”THANKS, THAT IS ALL”
>80 PRINT ”MAYBE SOME OTHER TIME” : END
>100 PRINT ”WRONG, TRY AGAIN”: GOTO 50
>200 REM TIME IS UP; CORRECT C STACK
>210 PRINT
>220 CLEARS : PRINT ”YOU LOST:YOU TOOK TOO MUCH TIME”
>230 CLEARI : REENABLE INTERRUPT
>240 GOTO 10
```

5.8.3 INIT

Statement	`INIT [integer]`
Mode	COMMAND and/or RUN
Type	Control

Description
INIT is used to initialize the bidirectional serial interface.

Note:

Reinitialization of the bidirectional interface in Command mode may make it impossible for the terminal to address the module due to discrepancies in the parameters.

Baud rate:	1:	110 Bd	
	2:	200 Bd	
	3:	300 Bd	
	4:	600 Bd	
	5:	1200 Bd	
	6:	2400Bd	
	7:	4800 Bd	
	8:	9600 Bd	(default)
Parity:	0:	even	(default)
	1:	odd	
	2:	mark ”1”	
	3:	space ”0”	
	4:	no parity evaluation	

Note:

If the parity is not to be evaluated although a data format with parity was specified, the parity on the transmitter side is always ZERO.

Handshake:	0:	No handshaking (default)
	1:	Evaluation of modem signals RS232C (V.24) only)
	2:	X-ON/X-OFF protocol
	3:	Programmer protocol
Data format:	0:	1 start bit, 7 data bits, 1 parity bit, 2 stop bits
	1:	1 start bit, 8 data bits, 1 parity bit, 1 stop bit (default)
	2:	1start bit, 8 data bits, 2 stop bits
	3:	1 start bit, 7 data bits, 2 stop bits
	4:	1 start bit, 7 data bits, 1 parity bit, 1 stop bit
	5:	1 start bit, 8 data bits, 1 stop bit
Interface:	0:	TTY
	1:	8RS232C (V.24) (default)

Note:

The serial interface must be initialized for 8 data bits for communication between programmer/PC and CP 521 BASIC. The interface parameters for the programmer/PC and the CP 521 BASIC must be identical.

Description: INIT initializes the bidirectional serial interface. The parameter list must always be specified in the form of a 5-digit number.

Example: **INIT 61241**

Note:

A value must be specified for the parity even when a data format without parity was chosen. In this case, the parity value has no effect.

The interface is initialized as follows:

- Baud rate: 2400
- Parity: odd
- Handshake: X-ON/X-OFF protocol activated
- Data format: 10-bit-charakter frame (1 start bit, 7 data bits, 1 parity bit, 1 stop bit)
- Interface: V.24

5.8.4 INIT#

Statement	`INIT# [integer]`
Mode	COMMAND and/or RUN
Type	Control

Description

INIT# is used to initialize the unidirectional interface.

Baud rate:	1:	110 Bd	
	2:	200 Bd	
	3:	300 Bd	
	4:	600 Bd	(default)
	5:	1200 Bd	
	6:	2400 Bd	
	7:	4800 Bd	

Parity:	0:	even	(default)
	1:	odd	
	2:	mark ”1”	
	3:	space ”0”	
Handshake:	0:	without BUSY signal	(default)
	1:	evaluation of BUSY signal	
Data format:	0:	1 start bit, 7 data bits, 1 parity bit, 2 stop bits	
	1:	1 start bit, 8 data bits, 1 parity bit, 1 stop bit (default)	
	2:	1 start bit, 8 data bits, 2 stop bits	
	3:	1 start bit, 7 data bits, 2 stop bits	
	4:	1 start bit, 7 data bits, 1 parity bit, 1 stop bit	
	5:	1 start bit, 8 data bits, 1 stop bit	

Description: INIT# initializes the unidirectional interface. The parameter list is specified in the form of a 4-digit number.

Nota:

The BUSY signal is active LOW.

Example: **INIT# 3010**

The interface is initialized as follows:

- Baud rate: 300
- Parity: even
- Handshake: BUSY signal is evaluated
- Data format: 11-bit character frame (1 start bit, 7 data bits, 1 parity bit, 2 stop bits)

5.8.5 BREAK

Statement	`BREAK ([integer])`
Mode	COMMAND and/or RUN
Type	Control

Description
BREAK can be used to control the interruptability of BASIC programs and output via <CTRL C>.
Following BREAK (0), programs can no longer be aborted with <CTRL C>.
Following execution of a BREAK (1) statement (default), the program can be aborted at any time.

Example

```
>10 STRING 1,5: A=1:  REM STRING DEFINITION
>20 $(0) = ”BREAK” : REM ”BREAK” IS THE PASSWORD
>30 BREAK (0): REM CONTROL-C DISABLE CONTROL
>40 FOR I=1 TO 1000 : REM DUMMY LOOP
>50 J=SIN (I)
>60 K=GET : IF K<>0 THEN 80 ELSE NEXT I
>70 END
>80 IF K=ASC ($(0),A) THEN A=A+1 ELSE A=1
>90 REM TEST FOR CONFORMANCE
>100 IF A=1 THEN NEXT I
>110 IF A=6 THEN 130 ELSE NEXT I
>120 END
>130 PRINT ”BREAK”
>140 BREAK (1) : REM ENABLE CONTROL-C
```

In the example above, entry of the word ”BREAK” interrupts program execution, i.e. the word ”BREAK” is a password.

Note:
You should provide for intervention in your program after BREAK (0) in order to be able to terminate the program, at least during the test phase. Failure to do so would mean that you would have to switch off the CP 521 BASIC if you wanted to abort the program.

5.8.6 DATA - READ - RESTORE

Statements	DATA - READ - RESTORE
Mode	RUN
Type	Assign

Description

Statement	DATA [expr] {, [expr], ...}

DATA specifies expressions that can be read in with a READ statement. In lines containing several such expressions, the expressions must be separated from one another by a comma.

Statement	READ [var] {, [var], ...}

READ reads the expressions specified in the DATA statement and assigns the values of the expressions to the variables in the READ statement. The READ statement must include one or more variables. If the READ statement includes several variables, they must be separated from one another by a comma.

Statement	RESTORE

RESTORE sets the internal read pointer to the beginning of the expressions specified in the DATA statement to enable them to be reread.

Example

```
>10 FOR I=1 TO 3
>20 READ A,B
>30 PRINT A,B
>40 NEXT I
>50 RESTORE
>60 READ A,B
>70 PRINT A,B
>80 DATA 10, 20, 10/2, 20/2, SIN(PI), COS(PI)
>RUN

   10 20
   5 10
   0 -1
   10 20
```

Special features
None

Description of the example:

Each time a READ statement is encountered, the next expression in the DATA statement is evaluated and assigned to the variables in the READ statement. DATA statements may be located anywhere in the program; they are not executed, nor do they produce errors of any kind. DATA statements are regarded as chains, and have the same effect as one single DATA statement. When the contents of DATA have been read in and another READ statement is encountered or there are no more DATA statements in the program, the program aborts and ERROR: NO DATA - IN LINE XX is displayed on the console.

5.8.7 DIM

Statement	`DIM [var(integer)]{,[var(integer)]}`
Mode	COMMAND and/or RUN
Type	Control

Description
DIM reserves space in memory for arrays (fields). Initially, the space required for arrays is assumed to be ZERO, i.e. none. In BASIC, arrays may have only one dimension, and the size of a dimensioned array may not exceed 254 array elements. Once variables have been dimensioned in a program, they may not be redimensioned. Any attempt to redimension an array is rejected with ERROR: ARRAY SIZE. If an array variable is used which was not assigned a dimension via a DIM statement, BASIC assumes a size of 10 elements for that array. Following execution of a RUN (5.6.1) or NEW (5.6.5) command or of a CLEAR statement (5.8.1), all arrays are reset to zero, i.e. removed from memory. The number of bytes reserved for an array is computed by adding one to the array size and multiplying the result with six. Array A (100) would thus require 606 bytes in memory. The maximum permissible number of dimensioned arrays is restricted only by the memory capacity.

Special features
Several variables can be dimensioned with a single DIM statement, e.g. DIM A(10), B(15), A1(20).

Example

```
ERROR DUE TO AN ATTEMPT TO REDIMENSION AN ARRAY

>10 A(5)=10     (BASIC ASSIGNS THE DEFAULT VALUE 10 AS ARRAY SIZE)
>20 DIM A(5)    (ARRAY CANNOT BE REDIMENSIONED)
>RUN

ERROR ARRAY SIZE - IN LINE 20

20    DIM A(5)
----------X
```

5.8.8 DO - UNTIL

Statement	`DO-UNTIL [rel expr]`
Mode	RUN
Type	Control

Description
DO-UNTIL [rel expr] statements make it possible to create loops in a BASIC program. All statements between DO and UNTIL [rel expr] are executed until the logical expression in the UNTIL statement is TRUE. DO - UNTIL loops may be nested.
Premature exiting of a DO-UNTIL loop in the main program may result in the error message "C-STACK-ERROR". If this occurs in subprograms, no error message will be output.

Example

```
SIMPLE DO-UNTIL

>10 A=0
>20 DO
>30 A=A+1
>40 PRINT A
>50 UNTIL A=4
>60 PRINT ”EOL”
>RUN

 1
 2
 3
 4
EOL

READY
>
```

```
NESTED DO-UNTIL

>10 DO : A=A+1 : DO : B=B+1
>20 PRINT A,B,A*B
>30 UNTIL B=3
>40 B=0
>50 UNTIL A=3
>RUN

 1  1  1
 1  2  2
 1  3  3
 2  1  2
 2  2  4
 2  3  6
 3  1  3
 3  2  6
 3  3  9

READY
>
```

Special features
None

5.8.9 DO - WHILE

Statement	DO-WHILE [rel expr]
Mode	RUN
Type	Control

Description
DO-WHILE [rel expr] statements are used to create loops in a BASIC program. DO-WHILE [rel expr] is the same as DO-UNTIL [rel expr] except that the statements between DO and WHILE [rel expr] are executed until the logical expression specified in the WHILE statement is TRUE. DO-WHILE and DO-UNTIL statements may be nested.
Premature exiting of a DO-WHILE loop in the main program may result in the error message "C-STACK-ERROR". If this occurs in subprograms, no error message will be output.

Examples

```
SIMPLE DO-WHILE

>10 DO
>20 A=A+1
>30 PRINT A
>40 WHILE A<4
>50 PRINT ”EOL”
>RUN

 1
 2
 3
 4
EOL

READY
>
```

```
NESTED DO-WHILE

>10 DO : A=A+1 : DO : B=B+1
>20 PRINT A,B,A*B
>30 WHILE B<>3
>40 B=0
>50 UNTIL A=3
>RUN

 1  1  1
 1  2  2
 1  3  3
 2  1  2
 2  2  4
 2  3  6
 3  1  3
 3  2  6
 3  3  9

READY
>
```

Special features
None

5.8.10 END

Statement	`END`
Mode	RUN
Type	Control

Description
END terminates the program run. Programs terminated with END cannot be resumed with CONT (5.6.2), and attempts to do so are flagged on the console with ERROR: CAN'T CONTINUE. If no END statement is written, the program run automatically terminates following execution of the last statement.

Examples

```
PROG. IS TERMINATED:

WHEN LAST STATEMENT HAS EXECUTED         WHEN END STATEMENT IS
                                         ENCOUNTERED

>10 FOR I=1 TO 4                         >10 FOR I=1 TO 4
>20 PRINT I                              >20 GOSUB 100
>30 NEXT I                               >30 NEXT I
>RUN                                     >40 END
                                         >100 PRINT I
1                                        >110 RETURN
2                                        >RUN
3
4                                         1
                                          2
READY                                     3
>                                         4

                                     READY
                                     >
```

Special features
None

5.8.11 FOR - TO - {STEP} - NEXT

Statements	`FOR-TO-{STEP}-NEXT`
Mode	COMMAND and/or RUN
Type	Control

Description
The FOR - TO - {STEP} - NEXT statements are used to create and control loops. Premature exiting of a FOR - TO - {STEP} - NEXT loop in the main program may result in the error message "C-STACK-ERROR". If this occurs in subprograms, no error message will be output.
Example

```
>10 FOR A=B TO C STEP D
>20 PRINT A
>30 NEXT A
```

The PRINT statement (5.8.22) on line 20 is executed six times if B=0, C=10 and D=2. The following values are output for ”A”:
0, 2, 4, 6, 8, 10. ”A” identifies the name of the subscript or loop counter. The value of ”B” is the initial value of the subscript, the value of ”C” the final value of the subscript, and the value of ”D” the increment for the subscript. The STEP statement is optional. If STEP and ”D” are omitted, the increment defaults to 1. NEXT adds the value of ”D” to the subscript. The subscript is then compared to the value of ”C”, which is the subscript limiting value.
The statements between FOR and NEXT are executed as long as the subscript does not exceed the limiting value. The subscript can also be ”decremented” (for instance, FOR I=100 TO 1 STEP - 1).

Examples

```
>10 FOR I=1 TO 4
>20 PRINT I
>30 NEXT I
>RUN

 1
 2
 3
 4

READY
>
```

```
>10 FOR I=0 TO 8 STEP 2
>20 PRINT I
>30 NEXT I
>RUN

 0
 2
 4
 6
 8

READY
>
```

Description

The FOR - TO - {STEP} - NEXT statements can be executed in Command mode. This feature makes it possible e.g. to output memory areas with a short program designed specifically for this purpose, for instance FOR I=512 TO 560: PH0.XBY(I).: NEXT I.

NEXT may also be used without a variable, thus enabling programs such as the one in the example below:

Example

```
>10 FOR I = 1 TO 100
>20 PRINT I
>30 NEXT
```

In this example, the variable for the NEXT statement is always the one that was used in the last FOR statement.

5.8.12 GOSUB - RETURN

Statements	`GOSUB - RETURN`
Mode	RUN
Type	Control

Description

Statement	`GOSUB [ln num]`

GOSUB [ln num] initiates execution of the subroutine that begins at the line number specified by [ln num].

Statement	`RETURN`

This statement resumes program execution with the statement following GOSUB. The GOSUB - RETURN sequence may also be ”nested”, i.e. a subroutine invoked with GOSUB may itself contain a GOSUB invoking another subroutine.

Examples

```
SIMPLE SUBROUTINE

>10 FOR I=1 TO 5
>20 GOSUB 100
>30 NEXT I
>100 PRINT I
>110 RETURN
>RUN

 1
 2
 3
 4
 5

READY
>
```

```
NESTED SUBROUTINES

>10 FOR I=1 TO 3
>20 GOSUB 100
>30 NEXT I
>40 END
>100 PRINT I,
>110 GOSUB 200
>120 RETURN
>200 PRINT I*I
>210 RETURN
>RUN

 1  1
 2  4
 3  9

READY
>
```

5.8.13 GOTO

Statement	`GOTO [ln num]`
Mode	COMMAND and/or RUN
Type	Control

Description
GOTO [ln num] is used to resume the program at any line number that follows GOTO.

Example

```
50 GOTO 100
```

If the program contains a line with the line number 100, the program continues at that line. If there is not, ERROR: INVALID LINE NUMBER is output to the console.
In contrast to the RUN command (5.6.1), the GOTO statement, when executed in Command mode, does not reset the memory area for variables or the interrupts. If GOTO is executed in Command mode after a line has been edited, however, the memory area for variables and all interrupts invoked by BASIC are reset. This is necessary because the memory area for variables and the BASIC program itself are in the same RAM, and program editing thus corrupts the variables.

5.8.14 ON GOTO and ON GOSUB

Statement	`ON [expr] GOTO [ln num]{,[ln num], ... [ln num]}` `ON [expr] GOSUB [ln num]{,[ln num], ... [ln num]}`
Mode	RUN
Type	Control

Description
The value of the expression in ON / ON is the subscript for the list of line numbers in GOTO/GOSUB. The program continues with the statement on the line whose number is addressed by the subscript.

Example

```
>10 ON Q GOTO 100, 200, 300
```

The program would continue with line number 100 if Q were 0, with line number 200 if Q were 1, with line number 300 if Q were 3, and so on. All specifications for GOTO and GOSUB also apply to the ON statement. If Q is less than zero, a BAD ARGUMENT ERROR is flagged. If Q exceeds the highest line number in the GOTO or GOSUB argument list, a BAD SYNTAX error is flagged. The ON statement enables a ”conditional branch” within a program.

5.8.15 IF - THEN - ELSE

Satement	`IF-THEN-{ELSE}`
Mode	RUN
Type	Control

Description
The IF statement initiates an alternate branch.
The IF - THEN - ELSE statements normally have the following format:

[ln num] IF [rel expr] THEN valid statement ELSE valid statement

The following example illustrates this more clearly:

```
>10 IF A=100 THEN A=0 ELSE A=A+1
```

(IF) A=100 when line 10 is executed, (THEN) A is assigned the value 0. If A is not equal to 100, then A is assigned the value A+1. The GOTO statement may be omitted when IF is to branch to different line numbers. The two examples below have the same effect:

```
>20 IF INT(A)< 10 THEN GOTO 100 ELSE GOTO 200
>20 IF INT(A)< 10 THEN 100 ELSE 200
```

In addition, the THEN statement can also be replaced by any other valid statement as shown below:

```
>30 IF A<>10 THEN PRINT A ELSE 10
>30 IF A<>10 PRINT A ELSE 10
```

The ELSE statement may be omitted. The program then continues with the next statement.

Example

```
>20 IF A=10 THEN 40
>30 PRINT A
```

In this example, the program continues with line number 40 when A=10. If A is not equal to 10, the program continues with line number 30.

Note:

Typically, branches are unnecessary when the IF - THEN - ELSE statements are written on the same line as other statements.

Example

```
>10 IF A=B THEN C=A : A=A/2
>20 PRINT A
```

The remainder of the line is executed only when the result of A=B is TRUE. As regards the example, this means that the BASIC interpreter would set C=A if A=B, then it would set A=A/2. If A is not equal to B, the interpreter would execute only PRINT A, and would ignore the statements C=A : A=A/2. The same logical interpretation also applies for ELSE.

5.8.16 INPUT

Statement	`INPUT or !`
Mode	RUN
Type	Input/Output

Description
The INPUT statement allows you to input data over the console while the program is executing. Data can be assigned to one or more variables simultaneously with a single Input statement. The variables must be separated from one another by a comma.
"!" is the abbreviated form of INPUT.

Example

```
INPUT A,B
```

A question mark (?) on the console prompts the user to enter two numbers, separated from one another by a comma. The BASIC interpreter responds with TRY AGAIN if the user does not enter two numbers or if the data type is incorrect.

Example

```
>10 INPUT A,B
>20 PRINT A,B
>RUN

?1

TRY AGAIN

?1,2
 1  2

READY
```

The INPUT statement can be written in such a way that the ensuing prompt provides an exact description of the entry required. The prompt must be written in quotes (”) behind the word INPUT. When a command precedes the first variable in the input list, the question mark normally displayed as input prompt is suppressed.

Examples

```
>10 INPUT ”INPUT CHARACTER” A        >10 INPUT ”INPUT CHARACTER-” ,A
>20 PRINT SQR(A)                     >20 PRINT SQR(A)
>RUN                                 >RUN

 INPUT CHARACTER                     INPUT CHARACTER-100
 ?100                                   10
 10
```

An INPUT statement can also be used to assign strings. Strings are always terminated with a character return (CR). If a single INPUT statement is to be used for several strings, the BASIC interpreter prompts you with a question mark.

Examples

```
>10 STRING 1,15                      >10 STRING 2,15
>20 INPUT ”NAME: ”,$(0)              >20 INPUT ”NAMES: ”,$(0),$(1)
>30 PRINT ”HELLO ”,$(0)              >30 PRINT ”HELLO ”,$(0),”AND ”,$(1)
>RUN                                 >RUN

NAME: MARIA                          NAMES: KLAUS
HELLO MARIA                          ?ANNE
                                     HELLO KLAUS AND ANNE
READY
                                     READY
```

A single INPUT statement can also be used to assign both strings and variables.

Example

```
>10 STRING 1, 10
>20 INPUT ”NAME(CR), AGE”,$(0),A
>30 PRINT ”HELLO ”,$(0),”, YOU ARE ”,A,”YEARS OLD”
>RUN

NAME(CR), AGE - KLAUS
?15
HELLO KLAUS, YOU ARE 15 YEARS OLD

READY
>
```

5.8.17 LET

Statement	`LET [var] = [expr]`
Mode	COMMAND and/or RUN
Type	Assign

Description
LET allocates the value of an expression to a variable. Its general format is:

```
LET [var] = [expr]
```

Examples

```
>10 LET A = 10*SIN(3)/100
>20 LET A = A + 1
```

As is apparent, the equal sign ”=” in the LET statement is not actually an equate symbol, but rather a ”replacement operator”, and means that A is to be replaced by A plus one. The word LET is always optional, i.e.:

```
>10 LET A = 2: REM IS EQUAL TO A = 2
```

When LET is not actually written, it is regarded as implied. In this manual, LET identifies both the ”physical” LET and the implied LET.

The LET statement can also be used to allocate strings to variables, i.e.:

```
LET $(0)=”THIS IS A STRING”
LET $(1)=$(0)
```

Strings cannot be allocated until a STRING [expr],[expr] statement has been executed, as otherwise a MEMORY ALLOCATION ERROR is flagged.

5.8.18 ONERR

Statement	`ONERR [ln num]`
Mode	RUN
Type	Control

Description
ONERR allows you to intercept any arithmetic errors occurring during the program run. However, ONERR can ”intercept” only errors of type ARITH. OVERFLOW, ARITH. UNDERFLOW, DIVIDE BY ZERO and BAD ARGUMENT. If an arithmetic error occurs after the ONERR statement, the interpreter continues the program with the line that follows the ONERR statement. ONERR provides a simple method of intercepting errors caused by the entry of incorrect data after an INPUT statement.

ONERR enables you to ascertain the type of error involved. To do this, it is necessary to test external memory location 257(101H) after intercepting the error. The error codes are listed below:

```
ERROR CODE = 10 - DIVIDE BY ZERO
ERROR CODE = 20 - ARITH OVERFLOW
ERROR CODE = 30 - ARITH UNDERFLOW
ERROR CODE = 40 - BAD ARGUMENT
```

This memory location can be queried with XBY (257).

Extending the ONERR function
The ONERR function also enables you to determine the number of the program line where the error occurred using the XBY statement. The address of the line number where the error occurred is entered in the following bytes:

600 (high byte)
601 (low byte)

The error code is at address 257 (as with the previous versions of the CP 521 BASIC)

Example:
The example below explains the relationships. After A passes, a division by 0 results in line 1020.

```
>100 REM ”Test program for ONERR”
>110 ONERR 400                          In the event of an arithmetic
>200 INPUT A                            error, continue program at line
>210 B=9*8*7*6*5*4*3                    number 400
>220 GOTO 1000
>400 PRINT ”ERROR CODE = ”,XBY(257)      Error code = 10
>410 I = 256*XBY(600)+XBY(601)          Calculation of the line number
>420 PRINT ”ERROR in line ”,I           Error in line 1020
>430 END
>1000 FOR I=0 TO A
>1010 A = A - 1
>1020 C = B / A                         Division results in zero
>1030 PRINT C                           after A passes!
>1040 NEXT I
```

5.8.19 ONTIME

Statement	`ONTIME [expr], [ln num]`
Mode	RUN
Type	Control

Description
The ONTIME statement was implemented to compensate for the fact that timers/counters operate in a s time range while the BASIC interpreter processes a program line in milliseconds. ONTIME generates an interrupt when the value of the special TIME operator is at least equal to the value of the expression follwing the ONTIME statement. The comparison is always made between the integer portion of TIME and the integer portion of the expression. The interrupt forces a GOSUB (5.8.12) to the line number [ln num] which follows ”expr” in the ONTIME statement.

Because the ONTIME statement works with the special TIME operator, a CLOCK1 statement must be executed in order for ONTIME to be effective. If the CLOCK1 statement is omitted, the special TIME operator is not incremented and ONTIME cannot go into force.

Periodic interrupts can be generated by reexecuting the ONTIME statement in the interrupt routine.

Example

```
>10 TIME=0 : CLOCK1 : ONTIME 2,100 : DO
>20 WHILE TIME<10 : END
>100 PRINT ”TIMER INTERRUPT AT -”, TIME,”SECONDS”
>110 ONTIME TIME+2,100 : RETI
>RUN
 TIMER INTERRUPT AT- 2.045 SECONDS
 TIMER INTERRUPT AT- 4.045 SECONDS
 TIMER INTERRUPT AT- 6.045 SECONDS
 TIMER INTERRUPT AT- 8.045 SECONDS
 TIMER INTERRUPT AT- 10.045 SECONDS

READY
```

The fact that the time (TIME) shown is 45 ms behind the time at which the interrupt is to be generated may cause some confusion. The reason for this is that the terminal on which this example was printed had a baud rate of 4800 and needed approximately 45 ms to actually print out the text TIMER INTERRUPT AT -” ”.

Should this delay be unacceptable, the value of the TIME operator should be allocated to a variable at the beginning of the interrupt routine.

Example:

```
>10 TIME=0 : CLOCK1 : ONTIME 2,100 : D0
>20 WHILE TIME<10 : END
>100 A=TIME
>110 PRINT ”TIMER INTERRUPT AT -”,A,”SECONDS”
>120 ONTIME A+2,100 : RETI
>RUN

 TIMER INTERRUPT AT- 2 SECONDS
 TIMER INTERRUPT AT- 4 SECONDS
 TIMER INTERRUPT AT- 6 SECONDS
 TIMER INTERRUPT AT- 8 SECONDS
 TIMER INTERRUPT AT- 10 SECONDS

READY
```

If additional ONTIME interrupts are to be possible, the last statement in the ONTIME interrupt routine must be a ”RETI” statement. Following execution of the RETI statement, the interpreter continues the program at the location at which it was interrupted by the ONTIME interrupt.
The ONTIME statement is a feature not available in most other BASIC dialects. This powerful statement makes periodic querying of the TIME operator unnecessary.

ONTIME can be rescinded with CLEARI.

5.8.20 CLOCK1 and CLOCK0

Statements	`CLOCK1 and CLOCK0`
Mode	COMMAND and/or RUN
Type	Control

Description

Statement	`CLOCK1`

CLOCK1 enables the BASIC-resident timer. Following execution of a CLOCK1 statement, the special TIME operator is incremented every 5 ms. When TIME reaches 65,535.995, an overflow is flagged and TIME starts again at 0.
The TIME operator counts from 0 to 65,535.995 seconds. As soon as the count reaches 65,535.995 seconds, it goes back to zero. The interrupts associated with CLOCK1 cause programs executing under MCS BASIC control to execute at approximately 99.6% of the normal speed. Servicing of timer interrupts thus takes up only about 0.4% of the total CPU time. This extremely low figure is possible only because of the CPU's fast, effective interrupt handling capability.

Statements	`CLOCK0`

CLOCK0 (zero) stops the timer. Once CLOCK0 has executed, the TIME operator is no longer incremented. CLOCK0 is the only statement that can disable the timer.

Special features
None

In addition to the timer for time-controlled program execution, it is also possible to set and read out the date and time over the external clock module with SET-CLOCK and RDCLOCK (5.13.1/2). The user can choose between German and American format.

5.8.21 TIME

Function	TIME
Mode	COMMAND and/or RUN
Type	Control

Description
The TIME operator is used to read out and/or set the timer. TIME is 0 following a RESET. The CLOCK1 statement (5.8.20) starts the timer. As long as the timer is enabled, the special TIME operator is incremented every 5 ms. TIME is specified in seconds.
When TIME is assigned a value using a LET statement (i.e. TIME=100), only the integer portion of TIME is modified.

```
>CLOCK1        (Enables the timer)
>CLOCK0        (Disables the timer)
>PRINT TIME    (Shows TIME)
 3.315
>TIME = 0      (Sets TIME to 0)
>PRINT TIME    (Shows TIME)
 .315          (Only the integer portion of TIME is modified)
```

5.8.22 PRINT

Statement	`PRINT or P. or ? [List of expressions]`
Mode	COMMAND and/or RUN
Type	Input/Output

Description
The PRINT statement tells the BASIC interpreter to output data to the console. It is possible to output expressions, strings, constants or variables, or a combination of these. The arguments in the list must be separated from one another by a comma. When the list concludes with a comma, carriage return and line feed are suppressed.
”P.” and ”?” are abbreviated forms of PRINT.

Example

```
>PRINT 10*10, 3*3       >PRINT ”MCS BASIC”       >PRINT 5,1E3
 100  9                 MCS BASIC                 5  1000
```

The values are output side by side, each value being separated from the preceding value by two blanks. Only the characters for carriage return and line feed are output when the PRINT statement is written without an argument list.

Note:

The bidirectional interface can be initialized with INIT (5.8.3).

Note
Line feed and carriage return are carried out automatically following PRINT.

SPECIAL OUTPUT FORMATTING STATEMENTS

TAB ([expr])

TAB ([expr]) is used in the PRINT statement to position data. TAB ([expr]) specifies the position for the next value in the output list. The BASIC interpreter ignores TAB when the printer head or the cursor has already reached or has exceeded the specified position.

Example

```
>PRINT TAB(5),"X",TAB(10),"Y"
     X    Y
```

SPC ([expr])

SPC ([expr]) is used in the PRINT statement to tell the BASIC interpreter to output a specific number of blanks (spaces). The number of blanks is specified as argument in the SPC statement.

Example

```
>PRINT A,SPC(5),B
```

The example above tells the interpreter to output five spaces between A and B in addition to the two spaces normally used as separators.

CR

CR in a PRINT statement executes a carriage return but suppresses a line feed. This makes it possible to display different items of data on one and the same line.

Example

```
>10 FOR I=1 TO 1000
>20 PRINT I,CR,
>30 NEXT I
```

The example above displays the same line again and again with different data each time.

USING (special symbols)

USING defines the format in which values are to be represented. The interpreter ”stores” the format after executing a USING statement, and subsequently displays all data in that format until a new USING statement is encountered. It is therefore unnecessary to reiterate USING unless you want a different format. The abbreviation for USING is ”U”. The following options are available for the USING statement:

USING (Fx) - All numbers are output in floating-point format. The value x specifies the number of significant digits to be output. If x=0, the BASIC interpreter suppresses trailing zeros; the number of digits therefore depends on the number. The interpreter always outputs at least three significant digits, even when x=1 or 2. The highest permissible value for x is 8.

Example

```
>10 PRINT USING(F3),1,2,3
>20 PRINT USING(F4),1,2,3
>30 PRINT USING(F5),1,2,3
>40 FOR I=10 TO 40 STEP 10
>50 PRINT I
>60 NEXT I
>RUN

 1.OO E 0   2.00 E 0   3.00 E 0
 1.000 E 0   2.000 E 0   3.000 E 0
 1.0000 E 0   2.0000 E 0   3.0000 E 0
 1.0000 E+1
 2.0000 E+1
 3.0000 E+1
 4.0000 E+1

READY
```

USING (#.#) - Tells the BASIC interpreter to output all numbers in integer or decimal format. The number of ”#” in front of the decimal point specifies the number of significant digits in the integer portion of the number. Only integer numbers are output if no decimal point is written.
USING (###.###), USING (######) and USING (######.##) are all valid statements. A USING statement may contain up to eight ”#” characters. If the BASIC interpreter is unable to output the value in the specified format (normally because it comprises too many digits), it displays a question mark (?) followed by the number.

Example

```
>10 PRINT USING(##.##),1,2,3
>20 FOR I=1 TO 120 STEP 20
>30 PRINT I
>40 NEXT I
>RUN

     1.00   2.00   3.00
     1.00
    21.00
    41.00
    51.00
    81.00
?101

READY
```

Note:
USING (Fx) and USING (#.#) always output the numbers in columns aligned to the decimal point for better readability.

USING (0) - This argument leaves the selection of the format up to the BASIC interpreter. The rule which the interpreter follows in making its selection is simple: Numbers between ±99999999 and ±0.1 are output in integer and decimal format, and all other numbers in USING (F0) format. Leading and trailing zero are suppressed. Following a RESET, the BASIC interpreter defaults to USING (0).

5.8.23 PRINT#

Statement	`PRINT# or P.# or ? # [List of expressions]`
Mode	COMMAND and/or RUN
Type	Input/Output

Description
PRINT#, P.# and ? # are identical to PRINT, P. and ? except that they route the data to the unidirectional interface instead of to the console. The descriptions of the PRINT, P. and ? statements also appply to PRINT#, P.# and ? #. P.# and # are abbreviations for PRINT#.

Note:
The unidirectional interface can be initialized with INIT# [integer] (5.8.4).

5.8.24 PH0., PH1., PH0.#, PH1.#

Statement	`PH0., PH1., PH0.#, PH1.#`
Mode	COMMAND and/or RUN
Type	Input/Output

Description

PH0. and PH1. are the same as PRINT except that they output the values in hexadecimal format. PH0. suppresses two leading zeros in numbers lower than 255 (0FFH). PH1. always outputs four hexadecimal digits. Values output with PH0. and PH1. are always followed by an ”H”, and are always integers (decimal places are truncated). The BASIC interpreter outputs numbers that are not within the permissible range for integers (which is from 0 to 65565 (0FFFFH), inclusive) in ”standard” format, leaving out the ”H”. Since integers can be entered in either decimal or hexadecimal format, PRINT, PH0. and PH1. can also be used to convert from decimal to hexadecimal or vice versa. The information presented in the description of the PRINT statement also applies to the PH0. and PH1. statements. PH0.# and PH1.# are the same as PH0. and PH1. except that they route their output data to the unidirectional interface instead of to the console.

Example

```
>PH0.2*2          >PH1.2*2          >PRINT 99H        >PH0.100
 04H               0004H             153               64H

>PH0.1000         >PH1.1000         >P.3E8H           PH0.PI
 3E8H              03E8H            1000               03H
```

5.8.25 PUSH

Statement	`PUSH [expr]`
Mode	COMMAND and/or RUN
Type	Assign

Description
The arithmetic expression or expressions listed in the PUSH statement are computed and then pushed onto the argument stack in succession. In conjunction with the POP statement (5.8.26), PUSH provides an easy facility for forwarding parameters to BASIC subroutines or swapping variables. The last value forwarded to the argument stack with PUSH is the first value read out of the stack with POP.

Special features
Several expressions can be entered in the argument stack with a single PUSH statement. To do this, simply list the expressions in the PUSH statement, separating each from its predecessor by a comma, i.e. PUSH [expr],[xpr] to [expr]. The last expression [expr] in the list is pushed onto the stack last.

Example

```
SWAPPING VARIABLES

>10 A=10
>20 B=20
>30 PRINT A,B
>40 PUSH A,B
>50 POP A,B
>60 PRINT A,B
>RUN

10  20
20  10

READY
>
```

```
FORWARDING PARAMETERS TO SUBROUTINES

>10 PUSH 1,3,2
>20 GOSUB 100
>30 POP R1,R2
>40 PRINT R1,R2
>50 END
>100 REM EQUATE: A=2,B=3,C=1
>110 POP A,B,C
>120 PUSH (-B+SQR(B*B-4*A*C))/(2*A)
>130 PUSH (-B-SQR(B*B-4*A*C))/(2*A)
>140 RETURN
>RUN

-1 -.5

READY
>
```

5.8.26 POP

Statement	`POP [var]`
Mode	COMMAND and/or RUN
Type	Assign

Description
The last value entered in the argument stack is assigned to the variable specified in the POP statement and the argument stack is subsequently popped (i.e. incremented by 6). Values can be entered in the stack with PUSH.

Note:
Attempts to execute a POP statement when the argument stack is empty are rejected with an A-STACK ERROR.

Special features
Several variables can be read out of the argument stack with a single POP statement. The variables in the argument list must be separated from one another by a comma (i.e. POP [var],[var] ... [var]).

Examples

Refer to the description of the PUSH statement

Note:

These powerful statements can be used to ”circumvent” the problem of global variables, a common problem in BASIC programs. This problem occurs because the ”main” program and all subroutines invoked in it have to use the same variable names (i.e. global variables). It is not always advisable to use the same variables in a subroutine that were used in the main program. As a result, variables are often reassigned (A=Q, for instance) before a GOSUB statement is executed. The problem of global variables can be avoided by reserving several variable names exclusively for subroutines (S1 and S2, for example) and forwarding them to the stack, as shown in the preceding example.

5.8.27 REM

Statement	REM
Mode	COMMAND and/or RUN
Type	Control (has no effect)

Description

REM is an abbreviation for REMark. This statement performs no operation, but it does allow the user to write commentary in his program. Commentary is normally needed to make a program more readable. In lines containing a REM statement, the portion of the line to the right of the word REM is interpreted as commentary, thus making it unnecessary to terminate a REM statement with a colon (:). A REM statement may, however, be preceded by a colon, thus allowing the programmer to write a comment on every program line.

Examples

```
>10 REM INPUT A VARIABLE
>20 INPUT A
>30 REM INPUT ANOTHER VARIABLE
>40 INPUT B
>50 REM MULTIPLY THE TWO VARIABLES
>60 Z=A*B
>70 REM OUTPUT THE RESULT
>80 PRINT Z


>10 INPUT A :REM INPUT A VARIABLE
>20 INPUT B :REM INPUT ANOTHER VARIABLE
>30 Z=A*B   :REM MULTIPLY THE TWO VARIABLES
>40 PRINT Z :REM OUTPUT THE RESULT
```

The following program cannot possible function correctly because the entire line is interpreted as commentary, which means that the PRINT statement will not be executed:

Examples

```
>10 REM OUTPUT THE VALUE : PRINT A
```

5.8.28 RETI

Statement	`RETI`
Mode	RUN
Type	Control

Description
RETI is used to reset an ONLINE interrupt. A new ONLINE interrupt cannot be serviced until RETI has executed. Subsequent interrupts will be ignored if the user fails to execute a RETI statement in the interrupt routine.

5.8.29 STOP

Statement	STOP
Mode	RUN
Type	Control

Description
STOP allows the programmer to stop the program run at specific points. Variables can be displayed and/or modified after a STOP. The program can be resumed with CONT. The STOP statement provides a simple facility for ”debugging” the program. For further details on the STOP - CONT sequence, refer to CONT (5.6.2).

Example

```
>10 FOR I=1 TO 100
>20 PRINT I
>30 STOP
>40 NEXT I
>RUN

 1
STOP  - IN LINE 40

READY
>CONT

 2
```

Note that the line number that is output following execution of a STOP statement is that of the statement that follows the STOP statement, not that of the STOP statement itself.

5.8.30 RROM

Statement	`RROM [integer]`
Mode	COMMAND and/or RUN
Type	Control

Description
RROM stands for RUN ROM. It selects a program from the memory submodule and starts that program. The integer written in the RROM statement must be a constant, and specifies the program that is to execute. In Command mode, RROM 2 is equivalent to entering ROM 2 and then RUN. Note, however, that RROM [integer] is a statement, which means that a program that is currently executing can force execution of an entirely different, memory submodule-resident program. This allows the programmer to make, as it were, a ”flying change” of programs.

If an invalid integer is entered in a RROM statement (invalid, for instance, because RROM 8 was entered but the submodule file contains only six programs, or because it contains no programs at all), the BASIC interpreter executes the statement following the RROM statement without flagging an error.
If the integer is outside the permissible range (1 to 254), however, an error is reported. An error is also reported if there is no memory submodule or if the submodule or its contents are unknown (no BASIC identifier).

Note:
Every time a RROM statement executes, all variables and strings are reset to zero, making it impossible to use RROM for forwarding variables or strings from one program to another. RROM also resets all interrupts generated by the BASIC interpreter.

5.9 Special Statements and Functions for Data Interchange

In addition to the standard commands (LIST, CONT, PRINT and so on), a number of auxiliary commands are provided expressly for the purpose of data interchange.

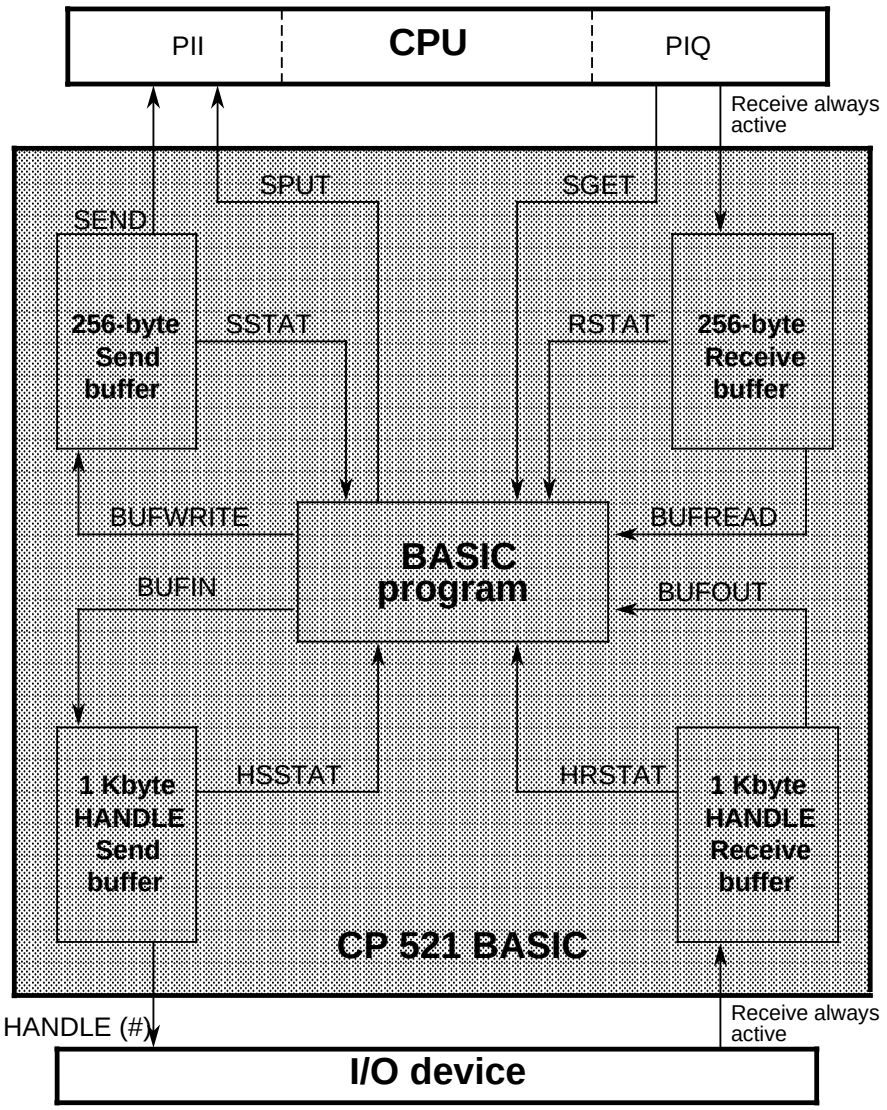


Fig. 5-7. Data Interchange Between CPU, CP 521 BASIC and I/O Device

Buffer comands

The module is equipped with the following four data buffers:

- Send buffer (256 bytes)
- Receive buffer (256 bytes)
- HANDLE Send buffer (1 Kbyte)
- HANDLE Receive buffer (1 Kbyte)

Send buffer

The SEND command (5.9.7) makes data available to the CPU. The CPU can fetch this data in the form of frames (CPU receive). A frame may comprise no more than 256 bytes.
You can display the status of the last Send request with SSTAT (5.9.3).
BUFWRITE (5.9.6) transfers data from a BASIC program variable to the Send buffer.

Receive buffer

Data from the CPU is received autonomously by a background task.
Data is transferred from the CPU's process output image (PIQ) to the CP 521 BASIC's 256-byte Receive buffer.
You can find out how many characters were entered in the Receive buffer and whether or not the Receive request has been serviced by invoking the RSTAT function (5.9.4).
BUFREAD (5.9.5) allocates a specific value from the Receive buffer to a BASIC program variable.

HANDLE Send buffer

The module provides a 1 Kbyte HANDLE Send buffer for forwarding data over the I/O interfaces. The HANDLE command (5.10.1) routes the data to the bidirectional interface. The data is transferred by a background task.
The HANDLE# command (5.10.1) outputs the contents of the HANDLE Send buffer to the unidirectional interface.

The BUFIN command (5.10.3) transfers the contents of a BASIC program variable to the HANDLE Send buffer.
The HSSTAT function (5.10.2) provides a numerical value representing the status of the last HANDLE Send request and the status of the interface.

HANDLE Receive buffer
The module provides a 1 Kbyte HANDLE Receive buffer for receiving data over the I/O interface. The Receive task is always active, and executes in the background.
The BUFOUT command (5.10.6) transfers data from the HANDLE Receive buffer to a BASIC program variable.
The HRSTAT function (5.10.4) provides a numerical value representing the status of the last HANDLE Receive request.

Note:

The data is transferred in interpretive mode, which means, for example, that a CTRL-C (03_H) is evaluated. A program abort can be prevented via a BREAK(0) statement.

5.9.1 SGET

Statement	`SGET [var],[var],[var],[var]`
Mode	RUN
Type	Input/Output

Description
The CP 521 BASIC reads four data words from the CPU's process output image (PIQ) and transfers them as number (integer) to the specified variable.
[var] must be the name of a valid numeric variable.
The PIQ is addressed by word.
The first word is mapped onto the first variable.

Example
SGET name1,name2,name3,name4
Four words forwarded by the CPU are stored in variables name1 to name4.

5.9.2 SPUT

Statement	`SPUT [expr],[expr],[expr],[expr]`
Mode	RUN
Type	Input/Output

Description
The CP 521 BASIC forwards four data words to the CPU's process input image (PII).
[expr] must specify a valid numeric expression.
The PII is addressed by word.
The first expression is mapped onto word 0 of the PII.

5.9.3 SSTAT

Function	`SSTAT`
Mode	RUN
Type	Input/Output

Description
Since only one SEND request is accepted at a time, you can use SSTAT to find out whether the last SEND request has been serviced. SSTAT provides a numerical value representing the status of the last Send request.
SSTAT returns the following status codes:

>0:	Send request still being serviced
0:	Send request has been serviced
-1:	S5 bus inactive (CPU stop)
-2:	No acknowledgement from CPU (time-out)
-3:	Errored acknowledgement from CPU

5.9.4 RSTAT

Function	`RSTAT`
Mode	RUN
Type	Input/Output

Description
After the Receive buffer has been partially or completely emptied with BUFREAD, RSTAT returns the number of characters remaining in the buffer in the form of a numeric value.
RSTAT returns the following:

>0:	Number of character still in the buffer
0:	No characters in the buffer
-1:	S5 bus inactive (CPU stop)
-2:	Request is being serviced
-3:	Request has been rejected

5.9.5 BUFREAD

Function	`BUFREAD [datatype,var{,length}]`
Mode	RUN
Type	Input/Output

`datatype`:	1:	Byte variable	(CPU format: 1 byte)
	2:	Integer variable	(16-bit fixed-point number: 2 bytes)
	3:	BCD variable	(value in BCD code: 2 bytes)
	4:	String variable	(”length” must be specified)

`var`: Valid variable name corresponding to data type

`length`: 1 to 254; number of characters to be transferred to the string. The parameter is required for type ”String” only.

Return values:	0:	Function executed without error
	1:	Function could not be correctly executed
	2:	Error during conversion of a BCD-coded number

Description
A value from the 256-byte Receive buffer is converted into the specified data type and transferred to the ”variable”. The ”length” parameter for a string variable must be in the range from 1 to 254. A ”1” is returned when

- ”length” <1 or >254
- the number of characters to be transferred exceeds the length of the string.
- the Receive buffer contains fewer characters than the number requested in BUFREAD.

A ”2” is returned when

- at least one digit of the BCD number to be converted is not within the permissible range (0 to 9).

5.9.6 BUFWRITE

Function	`BUFWRITE datatype,var{,length}`
Mode	RUN
Type	Input/Output

`datatype`:	1:	Byte variable	(CPU format: 1 byte)
	2:	Integer variable	(16-bit fixed-point number: 2 bytes)
	3:	BCD variable	(value in BCD code: 2 bytes)
	4:	String variable	(”length” must be specified)

`var`: Valid variable name corresponding to data type

`length`: 1 to 254; the number of characters to be transferred from the string variable. This parameter is required for type ”String” only.

Return values:	0:	The function was executed without error
	1:	The function could not be correctly executed
	2:	A digit is not within the permissible range

Description
The value of the specified variable is converted into ”datatype” and transferred to the Send buffer. A length of from 1 to 254 is permitted for variables of type ”String”.
A ”1” is returned when
1. the Send buffer cannot accommodate all the data
2. the string contains fewer characters than specified in the ”length” parameter.

The SEND statement (5.9.7) actually enables the data transfer.

5.9.7 SEND

Statement	`SEND [expr]`
Mode	RUN
Type	Input/Output

Description
The SEND command, with its frame length parameter [expr], is used to make data available to the CPU. A frame may comprise no more than 256 bytes. The CP 521 BASIC handles the blocking of the frames and handshaking with the CPU (refer to the Note below).
[expr] must be in the range from 1 to 256.

Note
The CPU fetches the frames in 8-byte subframes (6 bytes of useful data and 2 bytes of status info) from the module's Send buffer. When the CPU is ready to receive data, it informs the CP 521 BASIC by entering a synchronization value in byte 0 of the PIQ. The CPU acknowledges receipt of the data (Section 6).
Only one Send request can be issued at a time. The CP 521 BASIC ignores further requests without flagging an error.
You can query the status of the Send request with SSTAT and react accordingly.

5.10 Special Statements and Functions for Data Interchange Over the Serial Interfaces

A HANDLE Send buffer and a HANDLE Receive buffer are used for interchanging data over the serial interfaces. Both are circular buffers, and each has a 1 Kbyte capacity. Incoming data is always buffered. Send data can be edited in the Send buffer prior to transfer.
The following BASIC commands support data interchange over the serial interfaces:

- **Interface initialization**

- INIT [parameter]	Initializes the bidirectional interface
- INIT# [parameter]	Initializes the unidirectional interface

- **Forwarding data over the bidirectional interface**

- HANDLE [length]	Initiates the output of ”length” characters from the HANDLE Send buffer to the I/O device connected to the bidirectional interface
- HSSTAT	Status of the last Send request
- BUFIN [strvar(, length)]	Transfers the contents of ”strvar” to the HANDLE Send buffer

- **Forwarding data over the unidirectional interface**

- HANDLE # [length]	Same as HANDLE [length] but for the unidirectional interface
-HSSTAT	Status of the last Send request

- **Receiving data over the bidirectional interface**

- HRSTAT	Status of the Receive buffer. The module handles receipt of the data. The user need only query the status of the Receive buffer.
- GET	Reads a character from the HANDLE Receive buffer
-BUFOUT [strvar(, length)]	Transfers data from the HANDLE Receive buffer to a string with the name ”strvar”

5.10.1 HANDLE , HANDLE#

Statement	`HANDLE [expr], HANDLE# [expr]`
Mode	RUN
Type	Input/Output

Description (HANDLE)
HANDLE forwards the number of characters specified in ”expr” from the HANDLE Send buffer to the ***bidirectional*** interface.
HANDLE is a Send request, and is serviced by a background task.
The following values may be specified for ”expr”:
0< ”expr” 1024: Forward ”expr” characters from the HANDLE Send buffer
-1: Output all characters found in the Send buffer.

Description (HANDLE#)
HANDLE# outputs the number of characters specified in ”expr” from the HANDLE Send buffer to the **unidirectional** interface.
HANDLE# is a Send request, and is executed by a background task.
The following values may be specified for ”expr”:
0< ”expr” 1024: Transfer ”expr” characters from the HANDLE Send buffer
-1: Output all characters found in the Send buffer.

Note:

HANDLE and HANDLE# read the Send data from the same HANDLE buffer.
To avoid a conflict, only one Send request is serviced at a time.
The user can invoke HSSTAT to find out whether or not the previous request has been serviced.

5.10.2 HSSTAT

Function	HSSTAT
Mode	RUN
Type	Input/Output

Description
HSSTAT returns a numerical value representing the status of the last HANDLE Send request.
HSSTAT returns the following values:

- >0: Send request is still being serviced
- 0: Ready to accept a new Send request
- - 1: I/O device on unidirectional interface inactive: BUSY signal active for more than 20 seconds and user has specified evaluation of the BUSY signal
- - 2: Device on bidirectional interface inactive:
  - If XON/XOFF protocol specified: more than 20 seconds have elapsed since XOFF was last received
  - If modem signals specified: DTR=OFF or CTS=OFF for more than 20 s.

Note:

Negative return values are sometimes combinatory: - 3 means that both - 1 and -2 are applicable.

5.10.3 BUFIN

Function	`BUFIN [strvar(,expr)]`
Mode	RUN
Type	Input/Output

Description
The contents of the variable are copied to the HANDLE Send buffer. If the variable is a string variable, ”expr” (a value between 1 and 254) must specify the number of characters to be copied.
”var” must be a numeric variable, numeric constant or numeric expression.
A ”1” is returned if the buffer cannot accommodate all the data or if the string variable contains fewer characters than specified in ”expr”. In the latter case, a BAD ARGUMENT ERROR is reported and the function is not executed.
BUFIN returns the following values:
0: The function executed without error
1: The function could not be correctly executed.

Note:
The CHR() operator can be used to enter non-printable characters in the HANDLE Send buffer.

Example

```
20 STRING 1,40
30 $(0)=”CP 521 BASIC”
40  STATUS = BUFIN $(0),10
50 PRINT Status
60 HANDLE# 10

> RUN
  0     (BUFIN executed successfully)

(Output to printer (unidirectional interface)):

CP 521 BAS         (10 characters)
```

BUFIN CHR(x) or BUFIN CHR($(y),z) can be used to enter control characters in the HANDLE Send buffer.
CHR(x) and CHR($(y)z) must be used as described in the product manual (5.12.6).

Example

```
> 5  STRING 2,10
> 10 IF HSSTAT<>0 THEN 10
> 20 $(0)= ”ABCDEFG”
> 30 $(1)= ”1234567890”
> 40 X= BUFIN $(0),3
> 50 X= BUFIN CHR(13) : REM       ENTER CR
> 60 X= BUFIN CHR(10) : REM       ENTER LF
> 70 X= BUFIN $(1),10
> 60 HANDLE -1
> RUN
  ABC
  1234567890
```

5.10.4 HRSTAT

Function	HRSTAT
Mode	RUN
Type	Input/Output

Description
Unless there is an error, HRSTAT returns the number of characters in the HANDLE Receive buffer, i.e. the number of characters received but not yet fetched with GET or BUFOUT.
HRSTAT returns the following values:

>0: Number of characters in the HANDLE Receive buffer
0: The buffer contains no data
-1: Continuous break on the Receive circuit:
- If modem signals specified: DSR=OFF and/or CTS=OFF for more than 20 s
- If XON/XOFF protocol specified: more than 20 s have elapsed since the CP 521 BASIC sent the last XON, i.e. the module is no longer ready to receive.

-2: Parity error while receiving data over the bidirectional interface; HANDLE Receive buffer cleared
-4: More than 3 characters received after XOFF or DTR=OFF
-8: Receive buffer overflow

Errors -4 and -8 are reset when the HRSTAT function is invoked.

Note:

Negative return values are sometimes combinatory; -10, for instance, is used to flag both -2 and -8.

5.10.5 GET

Function	GET
Mode	COMMAND and/or RUN
Type	Input/Output

Description
GET reads and removes the first character from the HANDLE Receive buffer. The GET function returns the decimal equivalent of the character's ASCII code. A ”0” is returned if the buffer is empty.
In Command mode, the return value is always ”0”. The example below outputs the decimal equivalent of all characters entered on the keyboard.

Example

```
>10 REM *** GET function ***
>20 A = GET:IF A = 0 GOTO 20
>30 PRINT A,” (CHAR.”,CHR (A),” ENTERED ON KEYBOARD)”
>40 GOTO 20

65 (CHAR. "A" ENTERED ON KEYBOARD)
49 (CHAR. "1" ENTERED ON KEYBOARD)
24 (CHAR. "CONTROL-X" ENTERED ON KEYBOARD)
50 (CHAR. "2" ENTERED ON KEYBOARD)
```

5.10.6 BUFOUT

Function	`BUFOUT [strvar(,expr)]`
Mode	RUN
Type	Input/Output

Description
Starting with the first character in the HANDLE Receive buffer, characters are transferred to the string ”strvar”. ”expr” (which must be a number between 1 and 254) specifies the number of characters to be transferred.
”expr” must be a numeric constant, a numeric variable or a numeric expression.
BUFOUT returns the following values:
0: The function executed without error
1: The function could not be correctly executed
In the latter case, the function is not executed at all.
A ”1” is returned in the following cases:
”expr” < 1
”expr” > 254
”expr” > the number of characters in the buffer
”expr” > the length of the dimensioned string.

Note:

BUFOUT can be invoked in RUN mode only.

5.11 Arithmetic/Logical Operators and Expressions

5.11.1 Dyadic Operators

BASIC contains a complete set of arithmetic and logical operators. These operators are divided into two groups, i.e. dyadic operators for statements with two operands and unary operators for statements with only one operand. All statements for two operands have the following general format:

`[expr] OP [expr]`, where OP represents one of the following operators:

+ Operator for addition

Example

```
PRINT 3+2
5
```

/ Operator for division

Example

```
PRINT 100/5
20
```

** Operator for exponential functions

The second operand is the exponent for the first operand. The exponent for any number may not exceed 255.

Example

```
PRINT 2**3
8
```

* Operator for multiplication

Example

```
PRINT 3*3
9
```

- Operator für subtraction

Example

```
PRINT 9-6
3
```

.AND. Operator for logical AND

Example

```
PRINT 3.AND.2
2
```

.OR. Operator for logical OR

Example

```
PRINT 1.OR.4
5
```

.XOR. Operator for EXCLUSIVE OR

Example

```
PRINT 7.XOR.6
1
```

Logical operators .AND., .OR. and .XOR.

These operators logically combine the bit patterns of the relevant integers. Both arguments for these operators must be in the range from 0 to 65535 (0FFFFH). If they are not, the BASIC interpreter reports a BAD ARGUMENT ERROR. Non-integer values are truncated.

At first glance, the use of .OP. as the notational form for logical functions may seem somewhat unusual, but there is a reason for it. When the BASIC interpreter processes lines, it suppresses all blanks, but it inserts blanks in front of and behind all statements when user programs are output with LIST. However, the BASIC interpreter does not insert blanks in front of or behind operators.
In listings, lines such as 10 A=10 * 10 are output as 10 A=10*10, i.e. all of the blanks which the user entered before and after the operator are suppressed. If the notational form for logical operators were not .OP., a line that was entered as 10 B=A AND B would appear in the listing as 10 B=AANDB. Because periods are used to delimit logical operators, this line would appear in the listing as 10 B=A.AND.B, and would thus be easier to read.

5.11.2 Unary Operators - General Use

ABS`([expr])`

computes the absolute value of the argument.

Examples

```
PRINT ABS(5)            PRINT ABS(-5)
5                       5
```

NOT`([expr])`

computes the 16-bit one's complement of the argument. The argument must be a valid integer (i.e. a value between 0 and 65535 (0FFFFH), inclusive). Non-integers are truncated if necessary.

Examples

```
PRINT NOT(65000)        PRINT NOT(0)
535                     65535
```

INT`([expr])`

computes the integer portion of the argument.

Examples

```
PRINT INT(3.7)          PRINT INT(100.876)
3                       100
```

SGN`([expr])`

computes the value +1 for arguments greater than zero, the value 0 for arguments that are equal to zero, and the value - 1 for arguments less than zero.

Examples

```
PRINT SGN(52)          PRINT SGN(0)           PRINT SGN(-8)
1                      0                      -1
```

SQR`([expr])`

computes the square root of the argument. The argument may not be negative. Inaccuracies are limited to the least significant digit, and never exceed ±5.

Examples

```
PRINT SQR(9)           PRINT SQR(45)          PRINT SQR(100)
3                      6.7082035              10
```

RND

computes a pseudo random number in the range 0 to 1, inclusive. Using 16 bits as basis, RND generates 65536 pseudo random numbers before repeating the number sequence. As opposed to most BASIC dialects, the RND operator requires neither an argument nor a dummy argument. In fact, a BAD SYNTAX ERROR is reported when an argument is specified!

Example

```
PRINT RND
.30278477
```

PI

PI is not an operator, but a fixed constant with a value of 3.1415926.

5.11.3 Unary Operators - Logarithm Functions

LOG`([expr])`

computes the natural logarithm of the argument. The argument must be greater than zero. The result comprises seven significant digits.

Examples

```
PRINT LOG(12)          PRINT LOG(EXP(1))
2.484906               1
```

EXP`([expr])`

computes the value of the number ”e” (2.7182818) with the argument as exponent.

Examples

```
PRINT EXP(1)           PRINT EXP(LOG(2))
2.7182818              2
```

5.11.4 Unary Operators - Trigonometric Functions

SIN`([expr])`

computes the sine of the argument. The argument must be specified in radian measure. The result has seven significant digits. The argument must lie in the range ±200000.

Examples

```
PRINT SIN(PI/4)          PRINT SIN(0)
.7071067                 0
```

COS`([expr])`

computes the cosine of the argument. The argument must be specified in radian measure. The result has seven significant digits. The argument must lie in the range ±200000.

Examples

```
PRINT COS(PI/4)          PRINT COS(0)
.7071067                 1
```

TAN([expr])

computes the tangent of the argument. The argument must be specified in radian measure, and must be in the range ±200000. The result has seven significant digits.

Examples

```
PRINT TAN(PI/4)          PRINT TAN(0)
1                        0
```

ATN([expr])

computes the arc tangent of the argument. The result is in radian measure, and has seven significant digits. ATN returns a result between - PI/2 (3.1415926/2) and PI/2.

Examples

```
PRINT ATN(PI)            PRINT ATN(1)
1.2626272                .78539804
```

5.11.5 Operator Priorities

In mathematics, there are rules of priority which govern the order in which mathematical operations must be performed. These rules assign each operation to a specific hierarchical level. The example below shows why these rules are necessary:

4+3×2=?

Should the addition operation (4 + 3) be carried out first and then the result, i.e. ”7”, multiplied by 2, or should the multiplication operation (3×2) be carried out first and then 4 added to the result? According to the rules of mathematical hierarchy, multiplication takes priority over addition, which means first the multiplication of (3×2), then the addition of 4. The following therefore applies:

4+3×2=10

The mathemetical rules are simple. During the examination of an expression from left to right, no operations are carried out until an operator has a lower or equal priority. In the example above, the addition could not be performed because the multiplication has a higher priority. In BASIC, the operators have the following priorities:

1) Parenthesized expressions () / logical negations (NOT) and unary operators
2) Exponentiation (**)
3) Negation (-)
4) Multiplication (*) and division (/)
5) Addition (+) and subtraction (-)
6) Relational expressions (=, <>, >, >=, <, <=)
7) Logical AND (.AND.)
8) Logical OR (.OR.)
9) Exclusive OR (.XOR.)

Note:
The following general rule of thumb always applies, without regard to the rules of priority:
If you are not sure, use brackets.

5.11.6 Relational Expressions

Relational expressions contain the operators =, <>, >, >=, <, and <=. These operators are normally used to test the truth of a condition. In BASIC, the relational operators return a value of 65535 (0FFFFH) as result when the relational expression is TRUE, and a value of 0 when it is FALSE. The result is loaded into the argument stack, thus making it possible to display the result of a relational expression.

Example

```
PRINT 1=0        PRINT 1>0          PRINT A<>A       PRINT A=A
0                65535              0                65535
```

It may seem somewhat odd that a relational expression actually returns a result. The fact that it does has one very important advantage, which is that the .AND., .OR. and .XOR. operators can be combined, thus making it possible to test even a highly complex condition with a single statement.

Example

```
>10 IF A<B .AND. A>C .OR. A>D THEN
```

The NOT([expr]) operator can also be used in this way.

Example

```
>10 IF NOT (A>B). AND. A<C THEN
```

The combining, or ”gating”, of relational expressions with logic operators makes it possible to test highly complex conditions with a single statement. When using logic operators to gate relational expressions, care must be taken to observe the priorities of those operators. The logical operators were given a lower priority than the operators used in relational expressions only to enable the gating of relational expressions without using parentheses.

5.12 String Operations

A string comprises one or more characters, and are stored in memory. The characters stored in a string normally form a word or a sentence. Strings are easy to use because they allow the programmer to work with words instead of numbers. This makes for ”user-friendly” programs in which the programmer can, for example, identify objects or persons by name rather than by number.

BASIC supports one-dimensional string variables: The syntax of a string variable is $([expr]). The dimension of string variables (i.e. the value ([expr])) must be in the range from 0 to 254. This means that the programmer can define and process as many as 255 different strings. Initially, no space is reserved in memory for strings. The required space must be reserved with a STRING[number],[length] statement (5.12.1).

In BASIC, strings can be assigned in two different ways, i.e. with a LET statement and with an INPUT statement.
The operations and functions for processing strings are discussed in the following subsections.

5.12.1 STRING

Statement	`STRING [expr1],[expr2]`
Mode	COMMAND and/or RUN
Type	Control

Description
STRING[expr1],[expr2] reserves space in memory for strings (also refer to Section 5.12). Initially, no memory is reserved for strings. Attempts to define a string via a statement such as LET $(0)=”HELLO” before a memory areas for strings has been reserved are rejected with a MEMORY ALLOCATION ERROR.
In the statement STRING[expr1],[expr2], [expr1] specifies the total number of strings; the value range for [expr1] is from 0 to 254, inclusive. [expr2] specifies the maximum length of the strings, and must be in the range from 0 to 254, inclusive. These two values define the amount of space to be reserved for string variables.

Once a memory area has been reserved for strings, commands such as NEW or statements such as CLEAR cannot be used to release that area. There is only one way to release this area, and that is with the statement STRING 0,0. STRING 0,0 does not reserve a memory area for string variables.

Note:
In BASIC, the STRING [expr],[expr] statement has the same effect as the CLEAR statement. This is necessary because string variables and numeric variables reserve the same external memory area. For this reason, all variables are erased following execution of a STRING statement. Memory areas for strings should therefore be reserved as early as possible in a program (in the first statement, for instance), and the area for strings should not be released unless the programmer is willing to risk losing all his defined variables.

Example

```
>10 STRING 2,30
>20 $(0)=”THIS IS A STRING, ”
>30 INPUT ”WHAT IS YOUR NAME? - ”,$(1)
>40 PRINT $(0),$(1)
>RUN

WHAT IS YOUR NAME? - KLAUS

THIS IS A STRING, KLAUS
```

STRINGS can also be allocated to one another with a LET statement.

Example

```
$(1)=$(0)
```

The example above allocates the contents of STRING in $(0) to STRING $(1).

Note:

The STRING[expr[,[expr] statement also resets the ONXXXX statements. ONXXXX statements must therefore be written after the STRING statement.

5.12.2 ”+” (Combining Two Strings)

Function	”+”
Mode	COMMAND and/or RUN
Type	Input/Output

Description
The ”+” function combines two strings. Before executing this function, space must be reserved for strings $(0) to $(2) with ”STRING [expr],[expr]”.

Example

```
>10 STRING 3,12
>20 $(0)=”SIEMENS”          ($ (0) can accommodate 6 characters)
>30 $(1)=” AG”              ($ (1) can accommodate 3 characters)
>40 $(2)=$(0)+$(1)          ($ (2) can accommodate 10 characters)
```

The string variable $(2) now contains the string ”SIEMENS AG”.

Example

```
>10 STRING 3,7
>20 $(0)=”PETER ”           ($ (0) can accommodate 6 characters)
>30 $(1)=”MUELLER”          ($ (1) can accommodate 7 characters)
                            -------------------------

>40 $(2)=$(0)+$(1)          $ (2) would have to be able to
                            accommodate 13 characters
>50 PRINT $(2)
>RUN

 EXTRA IGNORED

 PETER M                    (Contents of $(2))
```

The warning EXTRA IGNORED is displayed on the monitor.

5.12.3 MID

Function	`MID (strvar, expr1, expr2)`
Mode	COMMAND and/or RUN
Type	Input/Output

Description
MID locates a substring in string variable ”strvar” that begins at position ”expr1” and is ”expr2” characters long. Counting begins at the left and starts with 1.

strvar: Valid string name, e.g. $(0)
expr1: Valid expression, 1 expr1 254
expr2: Valid expression, 1 expr1 254

Example

```
>5 STRING 2,25
>10 REM   12345678901234567890123
>20 $(0)=”Communications processor”
>30 $(1)= MID ($(0), 5, 6)
>40 PRINT $(1)
>RUN

unicat
```

String variable $(1) now contains the substring ”unicat”.
No error is flagged if the length of the substring exceeds the number of selectable characters (position+length>SLEN(strvar)), nor is an error flagged if the string variable to which the substring is to be transferred is too small to accommodate that substring; the maximum number of characters is transferred in the first instance, and as many characters as the string variable can accommodate in the second instance.

5.12.4 FIND

Function	`FIND strvar1, strvar2`
Mode	COMMAND and/or RUN
Type	Input/Output

Description
FIND returns the position of the first occurrence of substring ”strvar2” in string ”strvar1”.
FIND returns ”0” when:

- strvar2 is longer than strvar1 (SLEN(strvar2) > SLEN(strvar))
- strvar2 or strvar1 contains no characters (blank string)
- strvar2 cannot be found in strvar1

”strvar1” and ”strvar2” are valid string names.

Example

```
>5 STRING 2,40
 >10 REM            1         2         3         4
 >20 REM   1234567890123456789012345678901234567890
 >30 $(0)=”This is an example of the FIND function"
 >40 $(1)=”Example”
 >50 pos=FIND $(0), $(1)
 >60 PRINT”Found at position", pos
 >70 END

 >RUN
 Found at position 14
```

5.12.5 SLEN

Function	`SLEN strvar`
Mode	COMMAND and/or RUN
Type	Input/Output

Description
SLEN returns the length of string ”strvar”.
strvar: Valid string name
Return value: An integer value between 0 and 254

Example

```
>5 STRING 1,30
>10 REM            1         2
>20 REM   123456789012345678901 2
>30 $(0)="Programmable controller"
>40 SLAENGE=SLEN $(0)
>50 PRINT "The string length is",SLENGTH,"CHARACTERS"

>RUN
The string length is 22 characters
```

5.12.6 String Operators ASC and CHR

Function	`ASC(ASCII character) and CHR(expr)`
Mode	COMMAND and/or RUN
Type	Input/Output

BASIC provides two operators for manipulating strings. These two operators are called ASC() and CHR(), and their string processing capabilities are virtually unlimited.

Function	`ASC(ASCII character)`

Description
The ASCI() operator returns the decimal equivalent of the ASCII character specified in the parentheses.

Example

```
>PRINT ASC(A)
 65
```

65 is the decimal equivalent of the ASCII character ”A”. ASC() can also be used to find out whether an ASCII string contains a specific character.

Example

```
>5 STRING 1,20
>10 $(0)=”THIS IS A STRING”
>20 PRINT $(0)
>30 PRINT ASC($(0),1)
>RUN

 THIS IS A STRING
 68
```

Note:
In machine code, the end of a string is identified by a CR character. When using ASC, care must be taken not to overwrite or move this character, as this would make a string readout impossible or would truncate the string.
When using the ASC operator, always remember that there must not be a blank (space) between the operator name ”ASC” and the parenthesized expression ”()”.

In the form described above, $([expr]) specifies the string to be searched, whereas the expression following the comma specifies the character to be ”searched for”. In the preceding example, the first character in the string was selected, and the decimal equivalent for ASCII ”D” is 68.

Example

```
>5 STRING 1,50
>10 $ (0) = ”ABCDEFGHIJKL”
>20 FOR X = 1 TO 12
>30 ? ASC($(0),X), CHR($(0),X)
>40 NEXT X
>RUN

 65 A
 66 B
 67 C
 68 D
 69 E
 70 F
 71 G
 72 H
 73 I
 74 J
 75 K
 76 L
```

The numbers listed in the example above are the decimal equivalents of ASCII characters A through L.

ASC() can also be used to replace specific characters in a string.

Example

```
>5 STRING 1,15
>10 $(0)=”ABCDEFGHIJKL”
>20 PRINT $(0)
>30 ASC($(0),1)=75
>40 PRINT $(0)
>50 ASC($(0),2)=ASC($(0),3)
>60 PRINT $(0)
>RUN

 ABCDEFGHIJKL
 KBCDEFGHIJKL
 KCCDEFGHIJKL
```

ASC() allows the programmer to manipulate characters in strings. With a simple program, he can find out whether two strings are identical.

Example

```
>5 STRING 2,25
>10 $(0)=”SECRET”  REM THE PASSWORD IS SECRET
>20 INPUT ”WHAT IS THE PASSWORD - ”,$(1)
>30 FOR I=1 TO 6
>40 IF ASC($(0),I)=ASC($(1),I) THEN NEXT I ELSE 70
>50 PRINT ”YOU GUESSED IT”
>60 END
>70 PRINT ”WRONG, TRY AGAIN” : GOTO 20
>RUN

 WHAT IS THE PASSWORD - SECURE
 WRONG, TRY AGAIN
 WHAT IS THE PASSWORD - SECRET
 YOU GUESSED IT
```

Function	`CHR(expr)`

The ASC() operator's counterpart is CHR(). CHR() returns the ASCII equivalent of a numeric expression.

Example

```
>PRINT CHR(65)
A
```

As can ASC(), CHR() can ”pick out” characters from an ASCII string.

Example

```
>5 STRING 1,5
>10 $(0)=”BASIC”
>20 FOR I=1 TO 5 : PRINT CHR($(0),I), : NEXT I
>30 PRINT : FOR I=5 TO 1 STEP -1
>40 A=BUFIN CHR($(0),I), : NEXT I
>50 HANDLE -1
>RUN

BASIC
CISAB
```

In the example above, the parenthesized expressions that follow the CHR function have the same meanings as those which follow the ASC() operator.
In contrast to ASC(), however, CHR() cannot be assigned a value. A statement such as CHR($(0),1)=H, for instance, is impermissible, and is flagged with a BAD SYNTAX ERROR. Values in strings can be changed only with ASC().
CHR() may be used in a PRINT or BUFIN statement only.

Note:
When using CHR(), remember that there must not be a blank (space) between the operator name ”CHR” and the parenthesized expression ”()”.

5.13 Functions and Statements for Date and Time

5.13.1 SETCLOCK

Function	`SETCLOCK [strvar]`
Mode	COMMAND and/or RUN
Type	Control

Return value: Integer 0 or 1
0: Function executed without error
1: Function could not be properly

Description
SETCLOCK sets the module's hardware clock.
”strvar” (string constant or string variable) specifies the time, which may be given in one of the following two formats:

- German format:

	Hour :	Minute :	Second
Value range:	0 to 23	0 to 59	0 to 59

- American format:

	Time of day/	Hour/	Minute/	Second
Value range:	AM or PM	1 to 12	0 to 59	0 to 59

Example

```
>PRINT SETCLOCK ”AM/1/1/0”
0
READY
>
```

Sets the clock to ”1:01 AM”.

Note
The clock is not reset if the time specification is invalid or imcomplete, and SETCLOCK returns a ”1”.
The elements in the time specification may also be separated from one another by a space.

5.13.2 RDCLOCK

Statement	`RDCLOCK [strvar]`
Mode	COMMAND and/or RUN
Type	Control

Description
RDCLOCK returns the current time and date in string ”strvar” in the format (German/American) specified in the last SETCLOCK or SETDATE.

Note
If the string is too small to accommodate all of the time and date info, it is ”filled”, beginning at the left with the time, with as much of this info as possible.

Note:
The German format requires eight characters, the American format eleven.

5.13.3 SETDATE

Function	`SETDATE [strvar]`
Mode	COMMAND and/or RUN
Type	Control

Return value: Integer 0 or 1
0: The function executed without error
1: The function could not be executed without error

Description
SETDATE sets the date forwarded in the variable ”strvar” (string constant or string variable) in one of the following two formats:

- German format:

	Day :	Month :	Year
Value range:	1 to 31	1 to 12	(19)90 to (20)89

- American format:

	Month/	Day/	Year
Value range:	1 to 12	1 to 31	(19)90 to (20)89

Note
The date is not set if an imcomplete or invalid value is specified, and SETDATE returns a ”1”.
Each value in the date may comprise one or two digits, and the values may be separated from one another by a blank.
The year must be defined as two-digit number:
- 90 to 99 refers to the 20th century
- 0 to 89 refers to the 21st century

5.13.4 RDDATE

Statement	`RDDATE [strvar]`
Mode	COMMAND and/or RUN
Type	Control

Description
Following execution of the RDDATE statement, the string ”strvar” contains the date set on the module's hardware clock.
The date is in the same format (German or American) as that specified in the last SETCLOCK or SETDATE. The date always begins with the code for the day of the week, i.e.:

1: Sunday
2: Monday
3: Tuesday
4: Wednesday
5: Thursday
6: Friday
7: Saturday

Note
If the string is too small to accommodate all date info, it is ”filled” with as much of this info as possible.

Note:
Eleven characters are needed for the complete date, regardless of format.

5.13.5 T_ADJ

Statement	`T_ADJ [expr]`
Mode	COMMAND and/or RUN
Type	Control

Description
The system clock is adjusted at regular intervals to compensate e.g. for temperature-related clock errors. Correction values of from - 400 to +400, inclusive, are permitted, the base unit for this value being s/(30 days). The clock is adjusted at the rate of no more than one second per hour, so that there is no ”time jump”. The last correction factor specified is battery-backed. A positive correction factor slows the clock down, a negative factor speeds it up.

Special features
None

5.14 Miscellaneous Functions and System Control Parameters

5.14.1 Special Function Operators

Function	`XBY([expr])`
Mode	COMMAND and/or RUN
Type	Input/Output

Description
The XBY([expr]) function reads a value out of external RAM. The argument must be a valid integer (i.e. a value between 0 and 65535 ($0FFFF_H$)). An invalid argument is flagged as BAD ARGUMENT ERROR.

Example

```
A=XBY(0F000H)
```

In the example, variable A is set to the value in external RAM location $0F000_H$.

Note
XBY can be used to localize errors in the ONERR error recovery routine.

5.14.2 System Control Parameters

The system control parameters define or report on memory allocation.

MTOP

Following a RESET, BASIC formats the external store and sets the MTOP parameter to the highest valid RAM address, i.e. $7FFF_H$. BASIC does not use external RAM locations with addresses higher than the address in MTOP. Any attempt to set MTOP to a value that exceeds the highest valid RAM address is flagged with a MEMORY ALLOCATION ERROR.
Permissible value range: $0E00_H$ to $7FFF_H$

Examples

```
>PRINT MTOP
 2047

>MTOP=2000

>PRINT MTOP
 2000
```

LEN

The LEN system control parameter tells the user how much space (in bytes) the current program reserves. LEN is read-only, and as such cannot be assigned a value. LEN is set to ”1” for a ZERO program (i.e. no program). This byte is also the termination character for the program file.

FREE

The FREE system control parameter tells you how much RAM (in bytes) is available for variables/strings. The following equation applies for all programs currently in RAM.

FREE=MTOP - LEN - 3583

5.15 Error Messages and Special Features

5.15.1 Error Messages

BASIC has a relatively complex error message format. The general format for error messages in RUN mode is:

```
ERROR: XXX - IN LINE YYY
YYY BASIC STATEMENT
-------------X
```

XXX identifies the type of error, YYY the number of the program line on which the error occurred. For example:

```
ERROR: BAD SYNTAX - IN LINE 10
10 PRINT 34*21*
--------------X
```

X identifies the approximate line location within the line at which the error occurred. The X character may be off by one or two characters or expressions, depending on the type of error involved and where it occurred in the program. For errors that occur in Command mode, the type of error is output but the line number is not, as there are no line numbers in this mode. The following types of errors are flagged:

BAD SYNTAX
A BAD SYNTAX error is flagged when an invalid command, statement or operator is encountered and cannot be processed. The programmer should check his entries carefully for syntax errors, placing special emphasis on a search for the use of reserved words in variable names.

BAD ARGUMENT
A BAD ARGUMENT error is flagged when an argument is not within the permissible value range, e.g. T _ADJ=500.

ARITH. UNDERFLOW
An ARITH. UNDERFLOW error is flagged when the result of an arithmetic operation falls below the lower limiting value for a BASIC floating-point number. The lowest permissible floating-point number is ±1.0E - 127. ±1.0E - 80/1.0E + 80, for instance, would be flagged as an ARITH. UNDERFLOW error.

ARITH. OVERFLOW
An ARITH. OVERFLOW error is flagged when the result of an arithmetic operation exceeds the upper limiting value for a floating-point number. The highest permissible floating-point number is ±0.99999999E +127. 1.0E +70*1.0E + 57, for example, would be flagged as ARITH. OVERFLOW error.

DIVIDE BY ZERO
An attempt to divide by zero, e.g. 12/0, is flagged as DIVIDE BY ZERO error.

LINE TOO LONG
An audible signal warns the programmer when he attempts to enter more than 79 characters in a line.

NO DATA
The error message NO DATA - IN LINE XXX is output to the console when a READ statement was entered for which there was no DATA statement or when all data has already been read in and there is no RESTORE statement.

CAN'T CONTINUE
A program can be interrupted by entering CTRL-C on the console or by executing a STOP statement. Normally, the program can be resumed with CONT, but when the programmer edits the program after interrupting it and then enters CONT, a CAN'T CONTINUE error is flagged. The interpreter can execute a CONT command only when there was no editing between entry of CTRL-C or STOP and CONT.

A-STACK
An A-STACK (argument stack) error is flagged when the argument stack pointer goes out of range. This can happen when too many expressions are loaded into the stack with PUSH or when an attempt is made to read an empty stack with POP.

C-STACK
A C-STACK (control stack) error is flagged when the stack pointer goes out of range. The control stack reserves 158 bytes in external RAM. FOR-NEXT loops reserve 17 bytes in the control stack, DO-UNTIL, DO-WHILE and GOSUB loops 3 bytes each. This means that the BASIC interpreter can process a maximum of nine nested FOR-NEXT loops (9×7=153). A C-STACK error is flagged if the user attempts to exceed these limits. C-STACK errors are also reported when a RETURN precedes the associated GOSUB, a WHILE or UNTIL the associated DO, or a NEXT the associated FOR.

I-STACK
An I-STACK (internal stack) error is flagged when the interpreter does not have enough room in the stacks to process an expression.

ARRAY SIZE
An ARRAY SIZE error is flagged when, after dimensioning an array with DIM, the user attempts to address a variable that is not in that array.

Example

```
>DIM A(10)
>PRINT A(11)

ERROR ARRAY SIZE
READY
```

MEMORY ALLOCATION
A MEMORY ALLOCATION error is flagged when the user attempts to address strings ”outside” the defined string area. This error is also flagged when the MTOP parameter was assigned a value that is not a RAM address.

5.15.2 Special Features

The objective of this section is to help the user save time. The most prominent special features involve the method which BASIC uses to compress a program:

Note:
When entering an H variable or a keyword beginning with H after a line number, care must be taken that a space is entered between line number and H, as the interpreter will otherwise regard the line number as a hexadecimal number.

Examples

```
>20H=10   (INCORRECT)              >20 H=10   (correct)
>LIST                              >LIST
32   =10                           20    H=10
```

Note:
The argument for the ASC() operator may not contain blanks. Statements such as PRINT ASC( ) will therefore be flagged as BAD SYNTAX errors. Strings, however, may contain blanks, and statements such as LET $(0)=”HELLO, HOW ARE YOU” would thus pose no problem. The reason that ASC( ) would be flagged as BAD SYNTAX is that the BASIC interpreter suppresses all blanks when it processes a line, which would mean that ASC( ) would be stored as ASC(), and the interpreter regards this as incorrect.

Note

The German character format requires eight characters, the American character format eleven.

Interrupts

The BASIC interpreter can process timed interrupts. The statements for the interrupt routines are stored in the user program. The ONTIME statement (5.8.19) enables the interrupt.

6 Communications Interfaces

6.1 General Remarks . . . . . 6. - 1

6.2 Start-Up Procedures . . . . . 6 - 2

6.3 Data Interchange Between CPU and CP 521 BASIC . . . . 6 - 3
6.3.1 Sending Data Frames to the CP 521 BASIC . . . . . 6 - 5
6.3.2 Receiving Frames from the CP 521 BASIC . . . . . 6 - 18
6.3.3 Starting BASIC Programs . . . . . 6 - 26
6.3.4 Data Interchange Via Direct Access . . . . . 6 - 28
6.3.5 Set Clock (Status of the CP 521 BASIC and Current Clock Data) . . . . . 6 .- 29

6.4 Data Transfer Between CP 521 BASIC and I/O Device . . 6 - 34
6.4.1 The CP 521 BASIC's Bidirectional Interface . . . . . 6 - 34
6.4.2 The CP 521 BASIC's Unidirectional Serial Interface . . . . . 6 - 40

Figures

6-1. Sending and Receiving Data . . . . . 6 - 1
6-2. Sending Data Frames . . . . . 6 - 3
6-3. Receiving Data Frames . . . . . 6 - 4
6-4. Diagram for Sending Data . . . . . 6 - 7
6-5. Receiving Data . . . . . 6 - 19
6-6. Timing Diagram for Data Transfer Between CP 521 BASIC and I/O Device . . . . . 6 - 37
6-7. Sending Data over the Bidirectional Interface . . . . . 6 - 38
6-8. Receiving Data over the Bidirectional Interface . . . . . 6 - 39
6-9. Sending Data over the Unidirectional Interface . . . . . 6 - 40

Tables

6-1. CPU Requests for Sending Frames (PIQ) . . . . . 6 - 6
6-2. Coordination Request . . . . . 6 - 9
6-3. Coordination Info . . . . . 6 - 10
6-4. Data Transfer: Sending the 1st Subframe . . . . . 6 - 11
6-5. Acknowledgement for the 1st Subframe . . . . . 6 - 12
6-6. Data Transfer: Sending the 2nd Subframe . . . . . 6 - 13
6-7. Acknowledgement for the 2nd Subframe . . . . . 6 - 14
6-8. Data Transfer: Sending the 43rd Subframe . . . . . 6 - 15
6-9. Acknowledgement for the Last (43rd) Subframe . . . . . 6 - 16
6-10. Coordination Request . . . . . 6 - 17
6-11. Coordination Request . . . . . 6 - 20
6-12. Data Transfer: Receiving the 1st Data Block . . . . . 6 - 21
6-13. Acknowledgement for the 1st Subframe . . . . . 6 - 22
6-14. Data Transfer: Last (25th) Subframe Received . . . . . 6 - 23
6-15. Acknowledgement for the 25th Subframe . . . . . 6 - 24
6-16. Final Coordination Info . . . . . 6 - 25
6-17. CPU Request ”Start Program” . . . . . 6 - 27
6-18. Data from the Module to the CPU . . . . . 6 - 29
6-19. Module Status (Byte 0 in the PII) . . . . . 6 - 30
6-20. Date and Time . . . . . 6 - 31
6-21. Permissible Request (Byte 0 in the PIQ) . . . . . 6 - 33
6-22. Forwarding Additional Information in a ”Set Clock” Request . . . . . 6 - 33
6-23. RS232C (V.24) Control Signals in Modem Signal Mode . . 6 - 35

6 Communications Interfaces

This section deals with the theory of data interchange between the CPU, the CP 521 BASIC and the I/O device.
You will find a complete list of all FBs and the OB 1 required for sending data to and receiving data from the CP 521 BASIC in Section 8. Sections 6.3.1 and 6.3.2 are required reading only when information is required as regards troubleshooting or programming FBs.

6.1 General Remarks

Programmable drivers are used for communication between the CP 521 BASIC and I/O devices over the serial interfaces. Data is interchanged in interpretive mode with or without hardware or software protocols.
Data is interchanged in two stages:

1. CPU CP 521 BASIC
   Data is interchanged between the CPU and the CP 521 BASIC in 8-byte subframes (6 bytes of useful data).
2. CP 521 BASIC I/O devices
   Once data interchange over a serial interface has been initiated in the BASIC program, the CP 521 BASIC handles the interchange autonomously.

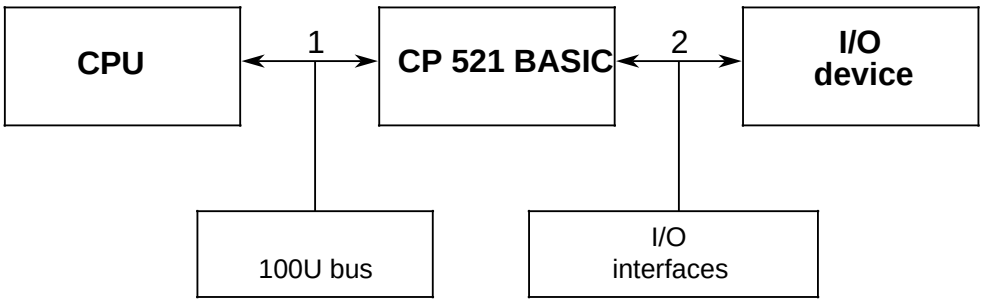


Figure 6-1. Sending and Receiving Data

The interchange depends on

- the type of interface (TTY or RS232C (V.24))
- and on whether or not handshaking was specified for the RS232C (V.24) interface (handshake ON or OFF).

6.2 Start-Up Procedures

The following prerequisites must be fulfilled before putting the module into operation:

1. Settings on the I/O device
The settings on the I/O device and the parameter initialization data for the CP 521 BASIC must be in conformance. For example, if your I/O device transfers data at 2400 baud, the module must also be set to 2400 baud.

2. Initializing the CP 521 BASIC
The module can be initialized in three different ways:

- On CP 521 BASIC runup, the Auto Baud routine sets defaults for the interface; the baud rate is automatically set to that of the I/O device.
- The CP 521 BASIC can be initialized with ”PROG” commands (5.7.4).
- The module can be initialized with INIT (5.8.3).

Note:

When the CP 521 BASIC is put into operation for the first time, it defaults to RS232C (V.24), 8 data bits, even parity, 1 stop bit, and no handshaking. The baud rate (110, 200, 300, 600, 1200, 2400, 4800, 9600) is set automatically to accord with that of the I/O device.

If you intend to use a **terminal or a programmer that has a TTY interface**, you must reinitialize the bidirectional interface (the default is RS232C (V.24)). It is recommended that you proceed as follows:

Connect a terminal or a programmer that has a RS232C (V.24) interface (PG 7XX).
Use INIT to change the interface parameters.
Store the parameters in battery-backed RAM with PROG1 or PROG2.
POWER OFF
Replace the terminal or programmer with the RS232C (V.24) interface with one that has a TTY interface.
POWER ON

6.3 Data Interchange Between CPU and CP 521 BASIC

SEND:

The CPU forwards frames (which may not exceed 256 bytes) in 8-byte subframes. Each subframe comprises 6 bytes of useful data and 2 coordination bytes. The CP 521 BASIC acknowledges the receipt of each subframe.

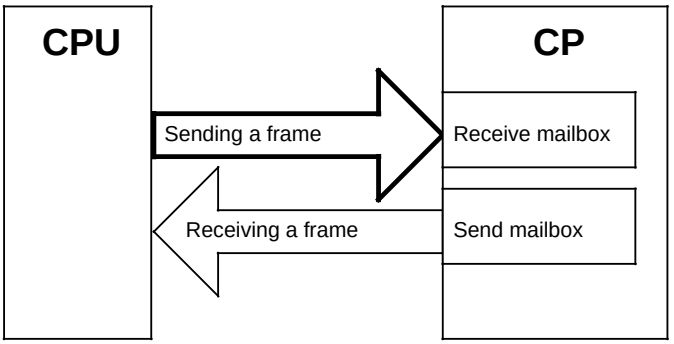


Figure 6-2. Sending Data Frames

1. CPU issues Request to Send

The CPU initiates the transfer of data to the CP 521 BASIC with a ”Request to Send” ($A001_H$).

2. CP Acknowledgement

The CP 521 BASIC acknowledges receipt of the request with 5001_H.

3. Subframes

The CPU evaluates the acknowledgement and, if no error was detected, begins forwarding data. The transfer begins with a $B0_H$ request and the number of the subframe, followed by six bytes of useful data. The last subframe need not necessarily contain six bytes of useful data, as it is the last ”fragment”.

4. CP acknowledgement
The CP 521 BASIC acknowledges the subframes, entering them in a Receive buffer as they come in until the frame is complete. A frame may comprise no more than 256 bytes.

5. Frame
The CP 521 BASIC combines the subframes into a frame. The frame is then available to the BASIC program for postprocessing.

RECEIVE:

A 256-byte Send buffer is provided for buffering data.
The data must be entered in the Send buffer by a BASIC program. Forwarding of the data from the CP 521 BASIC to the CPU is discussed below. The BASIC program must initiate the Send before the data can actually be transferred.

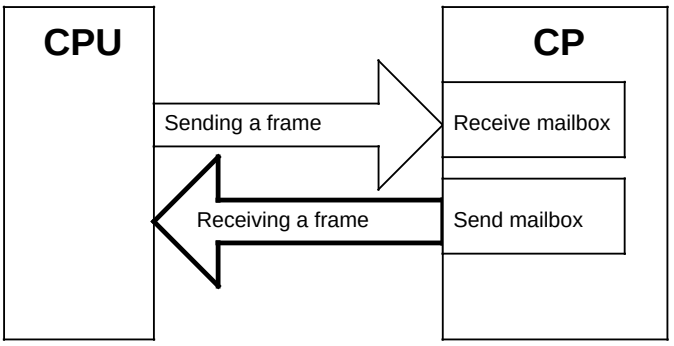


Figure 6-3. Receiving Data Frames

1. The CPU initiates the Receive
The CPU issues a ”Coordinate Receive Data” ($A080_H$) to the CP 521 BASIC.

2. CP acknowledgement
The CP 521 BASIC starts the data transfer, forwarding the first six bytes of the frame, together with the subframe number, from the Send buffer to the CPU (FIFO principle).

3. CPU acknowledgement
The CPU acknowledges the receipt of the data with $C0_H$ and the number of the subframe. The CP 521 BASIC then sends the next subframe, and so on, until the whole frame has been forwarded to the CPU.

4. Termination message
The CPU receives the final acknowledgements as soon as it issues a ”Coordinate Data Transfer” request:

- 5000_H in IW0 and $00XX_H$ in IW2 in response to requests $A001_H$ and $A080_H$ when a frame is complete and the data valid
- 5000_H in IW0 and $FFXX_H$ in IW2 in response to request $A001_H$ when the Receive buffer already contains a frame, the parameter for the frame length or end character is incorrect, or the data is invalid.
- 5000_H in IW0 and $FFXX_H$ in IW2 in response to request $A080_H$ when there is no frame in the Send buffer or the data is invalid.

6.3.1 Sending Data Frames to the CP 521 BASIC

The CPU forwards frames (which may not exceed 256 bytes) in 8-byte subframes. Each subframe comprises a two-byte header and six bytes of useful data.

1. The CPU initiates the data interchange by issuing request $A001_H$. This request also stipulates the frame size ( ).
2. The CP 521 BASIC acknowledges the request ( ).
3. The CPU evaluates the acknowledgement and, if it detects no errors, starts forwarding data. The transfer begins with $B0_H$ and a subframe number ( ), followed by six bytes of useful data. The last subframe may contain fewer than six bytes of useful data, depending on the frame length or end character.
4. The CP 521 BASIC enters the subframe in its Receive buffer until it has received the whole frame. The CP 521 BASIC acknowledges each subframe as it comes in ( ).
5. The CPU then forwards the next subframe to the CP 52 BASIC, and so on, until all subframes have been transferred. A complete frame may comprise no more than 256 bytes.

6. When it has received the last subframe, the CP 521 BASIC acknowledges with the termination message 5000_H.
7. The CP 521 BASIC combines the subframes into a frame. The data is then available to the BASIC program for postprocessing.

Note:

The CP 521 BASIC's Receive buffer can hold one frame. Only when the BASIC program has emptied the buffer can the CPU forward the next frame to the CP 521 BASIC.

Sending frames with length specification to the CP 521 BASIC

The information presented below describes the requests which the CPU issues when it wants the CP 521 BASIC to send data and the CP 521 BASIC's responses to these requests.

Table 6-1. CPU Requests for Sending Frames (PIQ)

Byte 0	Description
$A0_H$	Coordinate data transfer
$B0_H$	Data transfer CPU CP 521 BASIC

Data transfer via the CP 521 BASIC takes place in two stages:

- Coordination of data transfer between CPU and CP 521 BASIC
- Transfer of the data itself

The schematic diagrams on the next two pages illustrate the data transfer procedure. The diagrams are followed by descriptions of the steps involved in this procedure.

The diagrams illustrate the transfer of eight-byte subframes, alternating between the CPU request for a subframe and the CP 521 BASIC's acknowledgement.

Data transfer: CPU CP 521 BASIC

CPU CP 521 BASIC

Byte	0	1	2	3	4	5	6	7
	$A0_H$	01_H	nn_H	nn_H	nn_H	nn_H	xx_H	xx_H
→	Coord. request	Coord. byte for SEND (CBS	Frame size		End char.		Irrelevant	

CPU CP 521 BASIC

50_H	01_H	00_H	xx_H	nn_H	xx_H	xx_H	xx_H
Coord. info	Coord. byte	Data valid	Irrele-vant	No. of frames	Irrelevant		

CPU CP 521 BASIC

$B0_H$	01_H	nn_H	nn_H	nn_H	nn_H	nn_H	nn_H
Data trans.	Subframe No.	Data					

CPU CP 521 BASIC

50_H	01_H	00_H	01_H	nn_H	xx_H	xx_H	xx_H
Coord. info	Coord. byte	Data valid	Subframe No.	No. of frames	Irrelevant		

Figure 6-4. Diagram for Sending Data

CPU CP 521 BASIC

$B0_H$	02_H	nn_H	nn_H	nn_H	nn_H	nn_H	nn_H
Data trans.	Subframe No.	Data					

CPU CP 521 BASIC

50_H	01_H	00_H	02_H	nn_H	xx_H	xx_H	xx_H
Coord. info	Coord. byte	Data valid	Subframe No.	No. of frames	Irrelevant		

. . .

CPU CP 521 BASIC

$B0_H$	nr_H	nn_H	nn_H	nn_H	nn_H	nn_H	nn_H
Data trans.	Subframe No.	Last subframe					

CPU CP 521 BASIC

50_H	00_H	00_H	nr_H	nn_H	xx_H	xx_H	xx_H
Coord. info	Coord. byte	Data valid	Subframe No.	No. of frames	Irrelevant		

Figure 6-4. Diagram for Sending Data (continued)

Steps to are described in detail in the following.

CPU request: Coordinate data transfer ($A0_H$)

Request $A0_H$ is entered in byte 0 of the PIQ to coordinate data transfer. The entry in byte 0 determines the meaning of bytes 1 to 7.

Table 6-2. Coordination Request

Byte	Value	Description
0	$A0_H$	Request: Coordinate data transfer
1	01_H	Send frame
2	01_H	Send length (in this case 256 bytes), high-order byte
3	00_H	Send length (in this case 256 bytes), low-order byte
4	00_H	End characters 1, 2; in this case irrelevant
5	00_H	
6	Irrelevant	
7	Irrelevant	

Byte 1: The user gives permission to send by setting bit 0 to 1. This initiates the data transfer.

Bytes 2 and 3: Bytes 2 and 3 specify the frame size, which must be in the range from 0001_H to 0100_H.

When the frame size is 0, the CP 521 BASIC evaluates the end characters. The request is incorrect if both end characters and the frame size are 0.

When data is to be transferred from the CPU to the CP 521 BASIC, the ”frame size” or ”end character” parameter must be forwarded to the CP 521 BASIC along with the Send request.

CPU acknowledgement to request $A001_H$

When the CP 521 BASIC receives an $A0_H$ request (”Coordinate data transfer”), the Send bit is 01_H (”Send frame”) and no error is detected, the CP 521 BASIC acknowledges the CPU request as follows (PII):

Table 6-3. Coordination Info

Byte	Value	Description
0	50_H	Acknowledgement: Coordination info
1	01_H	Coordination bit ”Send” set
2	$00_H/FF_H$	Coordination data valid/invalid
3	Irrelevant	
4	$00_H/01_H$	No frame/frame in CP Send buffer
5	Irrelevant	
6	Irrelevant	
7	Irrelevant	

Byte 0: Acknowledgement to request: Coordinate Send.
The low-order nibble (X) may contain status or error info.

Byte 1: Coordination bit for ”Send” set. If CPU coordination data is invalid, the CP 521 BASIC sets the ”Send” bit to 0.

Byte 2: Indicates whether the coordination data is valid (00_H) or invalid (FF_H).

Byte 4: Indicates whether or not the CP 521 BASIC's Send buffer contains a frame.

As soon as the data transfer has been coordinated (steps and), the actual transfer begins with a $B0_H$ request (to).

Note:

Normally, the CP 521 BASIC forwards the date and time (6.3.5) to the CPU in bytes 1 to 7. If byte 0 is 50_H or 60_H, however, these values must not be interpreted as date/time info.

CPU request: "Send data" ($B0_H$)

After you have issued an $A0_H$ request and received the appropriate acknowledgement from the CP 521 BASIC, forward a $B001_H$ request with the first subframe (PIQ).

Table 6-4. Data Transfer: Sending the 1st Subframe

Byte	Value	Description
0	$B0_H$	Request: Send data
1	01_H	Number of the 1st subframe
2		Data
3		
4		
5		
6		
7		

Byte 0: Request: Send data. When it receives a $B0_H$ request, the CP 521 BASIC interprets bytes 2 to 7 as data.

Byte 1: You must increment the subframe number, whose initial value is 01_H, in your program for each subframe you forward to the CP 521 BASIC until the whole frame has been transferred.

CP acknowledgement to request $B001_H$

The CP 521 BASIC acknowledges a ”Send 1st subframe” request as follows (PII):

Table 6-5. Acknowledgement for the 1st Subframe

Byte	Value	Description
0	50_H	Acknowledgement to request: Send data
1	01_H	Coordination bit for ”Send” is set when data is valid
2	00_H	Data valid
3	01_H	Number of the 1st subframe
4	$00_H/01_H$	No frame/frame in CP Send buffer
5	Irrelevant	
6	Irrelevant	
7	Irrelevant	

Byte 3: Specifies the number of the subframe just received by the CP 521 BASIC (in this case 1).

Note:

Note that the CP 521 BASIC reacts to a signal change in byte 0 or byte 1 of the PIQ only. Since the $B0_H$ request does not change in byte 0 as long as subframes belonging to the same frame are being transferred, you must remember to increment the subframe number in byte 1 until all subframes in the frame have been transferred. If you send an incorrect subframe number, the data transfer is aborted with 5000_H and the CP 521 BASIC flags an ”invalid request” (40_H) in its status byte 0 for all subsequent $B0_H$ requests. The data is rejected.

If you issue the next $B0_H$ request after having increment byte 1 ( ), the CP 521 BASIC acknowledges by incrementing byte 3 ( ).

CPU request: $B002_H$

Table 6-6. Data Transfer: Sending the 2nd Subframe

Byte	Value	Description
0	$B0_H$	Request: Send data
1	02_H	Number of the second subframe
2		
3		
4		D a t a
5		
6		
7		

CP acknowledgement for request $B002_H$

Table 6-7. Acknowledgement for the 2nd Subframe

Byte	Value	Description
0	50_H	Acknowledgement to request: Send data
1	01_H	Coordination bit ”Send” is set when data is valid
2	00_H	Data valid
3	02_H	Number of the 2nd subframe
4	$00_H/01_H$	No frame/frame in the CP Send buffer
5	Irrelevant	
6	Irrelevant	
7	Irrelevant	

The last CPU request ( ) and the CP's acknowledgement to it ( ) are as follows when you send a variable-length frame of maximum length:

CPU request: $B02B_H$

Table 6-8. Data Transfer: Sending the 43rd Subframe

Byte	Value	Description	
0	$B0_H$	Request:	Send data
1	$2B_H$		Subframe number 43 (max.)
2		D a t a	
3		D a t a	
4		D a t a	(byte 255)
5		D a t a	(byte 256)
6			Irrelevant
7			Irrelevant

Note:

Since the maximum permissible frame size is 256 bytes, the maximum number of subframes into which a frame can be divided is 43 (42 6-byte subframes and one 4-byte subframe). There is thus room for only four bytes of data in the last subframe ($2B_H$).

CP acknowledgement to request B02B$_H$

Table 6-9. Acknowledgement for the Last (43rd) Subframe

Byte	Value	Description
0	50$_H$	Acknowledgement to request: Send data
1	00$_H$	Coordination bit for ”Send” is reset because this is the last subframe
2	00$_H$/FF$_H$	Coordination data valid/invalid
3	2B$_H$	Number of the 43rd (max.) subframe
4	00$_H$/01$_H$	No frame/frame in the CP Send buffer
5	Irrelevant	
6	Irrelevant	
7	Irrelevant	

Sending frames with end characters

The transfer of variable-length frames is similar to that of fixed-length frames, the difference being that you must forward one or two end characters in bytes 4 and 5 (Table 6-10) together with your A001$_H$ request (also see Send frame).

Table 6-10. Coordination Request

Byte	Value	Description
0	$A0_H$	Request: Coordinate data transfer
1	01_H	Send frame (Send bit)
2	00_H	Frame size (in this case variable)
3	00_H	Frame size (in this case variable)
4	$0D_H$	End character (only when frame size=0)
5	$0D_H$	End character (only when frame size=0)
6	Irrelevant	
7	Irrelevant	

Bytes 2 and 3: These bytes are used to specify the frame size (0000_H to 0100_H). If you want to send variable-length frames, set these bytes to 0000_H=variable frame size. You must then specify one or two end characters (bytes 4 and 5).

Bytes 4 and 5: Specify your end characters in these bytes. If you want to use two end characters, you must make entries in both of these bytes. If you want only one end character, enter it in byte 5.

Note:

If the frame exceeds 256 data bytes (> 0100_H), the CP 521 BASIC returns 5000_H and flags ”Invalid request” (40_H). The CP 521 BASIC flags this same error when it detects no end character(s) after receiving 256 bytes. The data forwarded by the CPU is rejected.

6.3.2 Receiving Frames from the CP 521 BASIC

The CP 521 BASIC handles the sending of frames to the CPU autonomously.
The CP 521 BASIC forwards data to the CPU from its 256-byte Send buffer.

You must program the CPU to fetch a frame from the CP 521 BASIC's Send mailbox in 8-byte subframes (two request bytes and six data bytes):

1. The CPU issues a ”**Coordinate Receive”** request to the CP 521 BASIC.
2. The CP 521 BASIC starts the data transfer, sending the first six bytes from its Send buffer (FIFO buffer).
3. The CPU acknowledges receipt of the data.
4. The CP 521 BASIC sends the next subframe; this process is repeated until the whole frame has been forwarded to the CPU.

The CPU immediately gets the final confirmation for the Receive frame when it issues the ”Coordinate Receive” request ($A080_H$).

Data interchange via the CP 521 BASIC takes place in two stages:

- First, data transfer between CPU and CP 521 BASIC must be coordinated.
- Only when this has been done can the CP 521 BASIC forward the data in its Send mailbox to the CPU.

Figure 6-8 shows how frames are received. The figure is followed by textual descriptions of the various steps.

The diagrams are based on 8-byte subframe, the CPU requests alternating with the CP 521 BASIC's acknowledgements.

Data transfer: CP 521 BASIC CPU

CPU CP 521 BASIC

Byte	0	1	2	3	4	5	6	7
	$A0_H$	80_H	xx_H	xx_H	xx_H	xx_H	xx_H	xx_H
→	Coord. request	Coord. byte for "REC."	Irrelevant					

CPU CP 521 BASIC

60_H	01_H	nn_H	nn_H	nn_H	nn_H	nn_H	nn_H
Data trans.	Sub-frame No.	Data					

←

CPU CP 521 BASIC

$C0_H$	01_H	xx_H	nn_H	nn_H	nn_H	nn_H	nn_H
Ack-nowl.	Sub-frame No.	Irrelevant					

→

. . .

CPU CP 521 BASIC

60_H	nr_H	nn_H	nn_H	nn_H	nn_H	nn_H	nn_H
Data trans.	Sub-frame No.	Data					

←

CPU CP 521 BASIC

$C0_H$	nr_H	xx_H	nn_H	nn_H	nn_H	nn_H	nn_H
Ack-nowl.	Sub-frame No.	Irrelevant					

→

. . .

CPU CP 521 BASIC

50_H	00_H	00_H	xx_H	nn_H	xx_H	xx_H	xx_H
Coord. info	Coord. byte	Data valid	Irrele-vant	No. of frames	Irrelevant		

←

Figure 6-5. Receiving Data

CPU request: Coordinate data transfer ($A0_H$)

You must enter request $A0_H$ in byte 0 of the PIQ to coordinate data transfer. This also defines the meaning of bytes 1 to 7.

Table 6-11. Coordination Request

Byte	Value	Description
0	$A0_H$	Request: Coordinate data transfer
1	80_H	Receive frame (Receive bit)
2	Irrelevant	
3	Irrelevant	
4	Irrelevant	
5	Irrelevant	
6	Irrelevant	
7	Irrelevant	

Byte 1: You grant permission to receive by setting bit 7 to 1. This initiates the data transfer. The CP 521 BASIC resets the Receive bit when a complete Receive frame has been transferred.

CP acknowledgement to request A080$_H$

When the CP 521 BASIC receives request A0$_H$ (Coordinate data transfer), the Receive bit (Receive frame) is set (80$_H$) and no errors are detected, the CP acknowledges the CPU request with 60$_H$ in byte 0:

Table 6-12. Data Transfer: Receiving the 1st Data Block

Byte	Value	Description
0	60$_H$	Data transfer CP 521 BASIC CPU
1	01$_H$	Number of the first subframe
2		
3		
4		D a t a
5		
6		
7		

Byte 1: Specifies the number of the subframe which the CP 521 BASIC forwards to the CPU. Beginning with subframe 01$_H$, the CP increments the subframe number each time it forwards a subframe to the CPU.

Byte 2 to 7: Data which the CP 521 BASIC transfers to the PII.

CPU acknowledgement ($C0_H$)

The CPU acknowledges the arrival of the subframe with $C0_H$ in byte 0:

Table 6-13. Acknowledgement for the 1st Subframe

Byte	Value	Description
0	$C0_H$	Acknowledges the arrival of the 1st subframe
1	01_H	Number of the first subframe
2	Irrelevant	
3	Irrelevant	
4	Irrelevant	
5	Irrelevant	
6	Irrelevant	
7	Irrelevant	

Byte 1: Number of the subframe received by the CPU (in this case 01_H). The subframe number in the acknowledgement must be identical to the number of the subframe that was transferred; if it is not, the data transfer is aborted.

Note:

The CP 521 BASIC transfers the same subframe again and again until it receives an acknowledgement from the CPU. In the data cycle in which the CPU acknowledges, it will therefore receive the subframe that it just acknowledged from the CP again. The CPU program must take this into account, as otherwise the CP 521 BASIC will receive an acknowledgement with the wrong subframe number and will therefore abort the data transfer.

The data transfer continues as described until the CPU has acknowledged the arrival of the last subframe ( ). The CP 521 BASIC then ends the transfer by sending a final confirmation ( ):

CP confirmation for CPU acknowledgement $C0_H$

Table 6-14. Data Transfer: Last (25th) Subframe Received

Byte	Value	Description
0	60_H	Data transfer CP 521 BASIC CPU
1	19_H	Number of the last subframe (in this case: 25_D)
2		
3		
4		D a t a
5		
6		
7		

CPU acknowledgement for the 25th subframe

Table 6-15. Acknowledgement for the 25th Subframe

Byte	Value	Description
0	$C0_H$	Acknowledges the arrival of the last subframe
1	19_H	Number of the 25th subframe
2	Irrelevant	
3	Irrelevant	
4	Irrelevant	
5	Irrelevant	
6	Irrelevant	
7	Irrelevant	

Confirmation message from the CP 521 BASIC

Table 6-16. Final Coordination Info

Byte	Value	Description
0	50_H	Coordination info
1	00_H	Coordination bit reset
2	00_H 01_H 02_H 81_H	Coordination data valid Error during data interchange between CP 521 BASIC and CPU No Receive frame Preceding Send request not yet serviced
3	Irrelevant	
4	00_H	Irrelevant
5	Irrelevant	
6	Irrelevant	
7	Irrelevant	

The CP 521 BASIC's final acknowledgement is retained until overwritten by the acknowledgement to a subsequent request (Example: After you have evaluated the CP 521 BASIC's final acknowledgement, write request 0000_H into word 0 on the CP to overwrite the final acknowledgement with the current data and time).

6.3.3 Starting BASIC Programs

There are several ways to start a program on the CP 521 BASIC:

- By entering the BASIC keyword RUN or RROM in Command mode.
- A BASIC program in RAM or on the external memory submodule is started when the mode selector is at RUN on power-up and parameter 1 or 2 was specified in the last PROG command.
  An automatic start can be initiated by defining a start program on the CP 521 BASIC with PROG1 or PROG2.
- A program can be started without regard to the position of the mode selector (PROG/RUN) by issuing a $D0_H$ request to the CPU.

The first two options were discussed in the preceding section.
The use of a CPU request to start a program is discussed in the next subsection. This method can be used, for example, to restart a BASIC program that was STOPPED because of a program error.
This CPU request cannot be used to interrupt a program that is already executing.

Peculiarities for RUN-/STOP requests
If the CPU request $D100_H$ (stop BASIC program) is transferred when the CP 521 BASIC is in the input mode (BASIC program is not in RUN mode), it will be stored. After starting the program with a RUN request (e.g. $D001_H$), the latter is aborted after processing of the first program line.
Likewise, any RUN request transferred when a BASIC program is set to RUN will also be stored. After a STOP request, the CP 521 BASIC will again enter RUN mode.
Issue the RUN/STOP request twice in order to make sure that the BASIC program has been started/stopped properly. The two requests must be separated by a 0000_H request.

CPU request ”Start program” ($D0_H$)

Table 6-17. CPU Request ”Start Program”

Byte	Value	Description
0	$D0_H$	”Start program” request
1	Nr.	Number of the program to be started
	00_H	Program is in RAM
	01_H to FE_H	Program is on memory submodule
	FF_H	Reset BASIC interpreter
2 to 7	Irrelevant	-

Byte 0: ”Start program” request
Only when ***no*** program is executing when the request is issued.

Byte 1: Contains the number of the program to be started.
Number=00_H: The program in the CP module's RAM is started. If there is no program in RAM, the interpreter enters the Command mode and waits for terminal entries.
Number=01_H to FE_H: The memory submodule-resident program with the specified number is started. If the memory submodule does not contain a program with this number, the interpreter enters the Command mode and waits for terminal entries.
Number=FF_H: The CP 521 BASIC is reset, which is equivalent to POWER ON.

Byte 2 to 7: Irrelevant

6.3.4 Data Interchange Via Direct Access

Requests $A0_H$ and $B0_H$ support communication between CPU and CP 521 BASIC. To interchange small amounts of data, it might be advisable to program your own protocol.
The BASIC interpreter supports this with the SGET and SPUT commands (5.9.1 and 5.9.2), which provide direct access to the CPU's transfer interface. These commands can be used to send/receive 8 bytes (4 words).
Should you write your own protocol, care must be taken never to enter the following reserved job request numbers in byte 0:

- $A0_H$
- 10_H

”Illegal request” (40_H) is returned when numbers > $0F_H$ (requests) are used.

The example below shows how special functions with parameters can be initiated via a user-written protocol.

STL	Description
: :L KH 0101 :T QW 120 :L KH 3E5F :T QW 122 :L KH 57E8 :T QW 124 :L KH 3A97 :T QW 126 :	 Write identifier for user-written protocol in: Word 0 Word 1 Word 2 Word 3

These four words can then be read out with SGET W0,...,W3.

Note:

A signal change is required in word 0 to fetch data or requests.

Note:

Use of the CP 521 BASIC in an ET 200
When using the CP 521 BASIC in connection with an ET 200, it is advisable to employ send and receive requests ($A001_H$ and $A080_H$ on CPU side, BUFREAD and BUFWRITE at CP 521 BASIC end). The acknowledgement procedure ensures safe data exchange between the CPU and CP 521 BASIC under any circumstances.

6.3.5 Set Clock (Status of the CP 521 BASIC and Current Clock Data)

The CP 521 BASIC makes information available cyclically via the PII:

Table 6-18. Data from the Module to the CPU

Byte	Description
0	Status of the CP 521 BASIC
1	Day of the week
2 to 7	Date and time set from the CP 521 BASIC's clock

This information can be read in the control program using Load operations and subsequently evaluated.

Read module status (byte 0 in the PII)

PII byte 0 (”Module status”) is subdivided into two nibbles, each containing different information which can be combined as required. The signal states of bits 4 to 7 (high-order nibble) of byte 0 are for the clock status.

Table 6-19. Module Status (Byte 0 in the PII)

Bits 4 to 7	Bits 0 to 3	Status	Description
1	X	Clock defect	
2	X	Default set	The clock was set to 1.1.90, 00.00.00.
3	X	Clock/date error	At least one of the specified values was out of range. The new data is rejected and the clock is not reset.

X: Signal state for the other nibble is irrelevant

Example: Querying the module status (byte 0 in the PII)

The module is plugged into slot 7 (start address 120).
Output Q 1.0 is to be set to signal a battery defect.

STL FB 100	Description

`NAME :ERROR1`
`:L IB 120`
`:L KH 000F`
`:AW`
`:`
`:`
	Read byte 0 (status byte) into ACCUM1. Word-oriented ANDing with $000F_H$ sets bits 4 to 7 to ”0”. The signal states of bits 0 to 3 are not affected.

`:L KH 0007`
`:><F`
`:BEC`
`:`
	Compare contents of ACCUM with 0007_H. If the status byte does not contain the value 7_H, the battery is OK and the block is exited.

`:= Q 1.0` `:BE`	Battery defect. Output Q 1.0 is set.

Date and time (bytes 1 to 7 in the PII)

The CP 521 BASIC makes the clock data available in BCD format. The low-order nibble of byte 1 contains the day of the week.

Table 6-20 . Date and Time

Byte	Value range	Description
1	1 to 7	Bits 0 to 3 : 1=Sunday, 2=Monday, 3=Tuesday, 4=Wednesday, 5=Thursday, 6=Friday, 7=Saturday
2	01_{BCD} to 31_{BCD}	Day
3	01_{BCD} to 12_{BCD}	Month
4	00_{BCD} to 99_{BCD}	Year
5	00_{BCD} to 23_{BCD} 01_{BCD} ... 12_{BCD} 81_{BCD} ... 92_{BCD}	Current hour: 24-hour mode 12-hour mode, a.m. 12-hour mode, p.m.
6	00_{BCD} to 59_{BCD}	Minute
7	00_{BCD} to 59_{BCD}	Second

Example: Reading out the current clock data

The module is plugged into slot 7 (start address 120).
The clock data is to be output over digital output modules beginning at address 8.

STL FB 100	Description

`NAME :CLKDATA`
`:L IB 121`
`:T QB 8`
`:L IW 122`
`:T QW 9`
`:L IW 124`
`:T QW 11`
`:L IW 126`
`:T QW 13`
`:BE`
	Read out the current clock data and transfer to the digital output modules

Note:

You can also evaluate the

- Clock defect ($1X_H$)
  Evaluation of this error serves a practical purpose only following a start or restart.
- Default set ($2X_H$)
  Evaluation of this error serves a practical purpose only when your CP 521 BASIC is equipped with a backup battery.
- Time or date error ($3X_H$)
  Evaluation of this error serves a practical purpose only after you have set the clock.

CPU request "Set clock" (10_H)

Table 6-21. Permissible Request (Byte 0 in the PIQ)

Byte 0	Byte 1	Description
10_H		Set clock, specify values in bytes 2 to 7

Specify the values in bytes 2 to 7 in BCD. The day of the week is computed from the specified date.
To retain a specific value, simply enter FF_H in the relevant byte.
The clock is not set if a value is out of range, and the CP 521 BASIC flags a "time/date error".

Table 6-22. Forwarding Additional Information in a "Set Clock" Request

Byte	Description	Value
2	Day	01_{BCD} to 31_{BCD}
3	Month	01_{BCD} to 12_{BCD}
4	Year*	00_{BCD} to 99_{BCD}
5	Hour	00_{BCD} to 23_{BCD} for 24-hour mode 01_{BCD} ... 12_{BCD} for 12-hour mode, a.m. 81_{BCD} ... 92_{BCD} for 12-hour mode, p.m.
6	Minute	00_{BCD} to 59_{BCD}
7	Second	00_{BCD} to 59_{BCD}

* Leap-year recognition is automatic

Note:

12-hour mode must be preset in the BASIC program (5.13.1).

Note:

You should execute a plausibility test on the following four bytes when you want to use the clock data to initiate actions:

Status	(Byte 0):	
Day of the week	(Byte 1):	01_{BCD} to 07_{BCD}
Day	(Byte 2):	01_{BCD} to 31_{BCD}
Month	(Byte 3):	01_{BCD} to 12_{BCD}

6.4 Data Transfer Between CP 521 BASIC and I/O Device

Data is interchanged over the CP 521 BASIC's bidirectional interface in full-duplex mode.

6.4.1 The CP 521 BASIC's Bidirectional Interface

The CP 521 BASIC's bidirectional interface can be initialized with INIT (5.8.3).

Bidirectional RS232C (V.24) interface in modem signal mode

The CP 521 BASIC's RS232C (V.24) interface can generate the following control signals in modem signal mode:

Table 6-23. RS232C (V.24) Control Signals in Modem Signal Mode

Control signal	State	Description
Outputs		
TxD		Send data The CP 521 BASIC holds the Send circuit at logical 1 when idle (V - 3 V).
DTR	 ON OFF	**D**ata **T**erminal **R**eady CP operational and ready to receive CP not operational, not ready to receive
RTS	 ON OFF	**R**equest **t**o **s**end CP in Transmit mode CP not transmitting
Inputs		
RxD		Receive data I/O device must hold Receive circuit to logical 1 (V - 3 V).
DSR	 ON OFF	**D**ata **s**et **r**eady I/O device operational and ready to receive I/O device not operational, not ready to receive
CTS	 ON OFF	**C**lear **t**o **s**end I/O device can receive characters from CP 521 BASIC CP 521 BASIC expects this as response to ”RTS”=”ON” I/O device cannot receive characters from CP 521 BASIC

In ”***Handshake OFF***” ***mode***, the CP 521 BASIC evalutes only the RxD circuit. When the CP 521 BASIC is not receiving data from the I/O device, the latter must hold the RxD circuit to logical ”1”.

Note:

Immediately after start-up, the CP 521 BASIC assumes that the ”communication partner” is ready to receive, even when no XON is transmitted.

Note:

The use of an XON/XOFF protocol is not possible when control signals are evaluated in ***”Handshake ON” mode***.

Data interchange between CP 521 BASIC and I/O device:
The CP 521 BASIC sets the ”DTR” output to ”ON” on start-up to show that it is operational and ready to receive data.

Example: The CP 521 BASIC wants to send data

1. The CP waits for ”DSR”=”ON”
   HHSTAT can be used to find out whether or not the I/O device set ”DSR” to ”ON” within 20 seconds.
2. The CP sets ”RTS” to ”ON”
3. The CP waits for ”CTS”=”ON”
   HHSTAT can be used to find out whether or not the I/O device set ”CTS” to ”ON” within 20 seconds.
4. The CP sends data
5. After sending the data, the CP sets ”RTS” to ”OFF”
6. The I/O device sets ”CTS” to ”OFF”

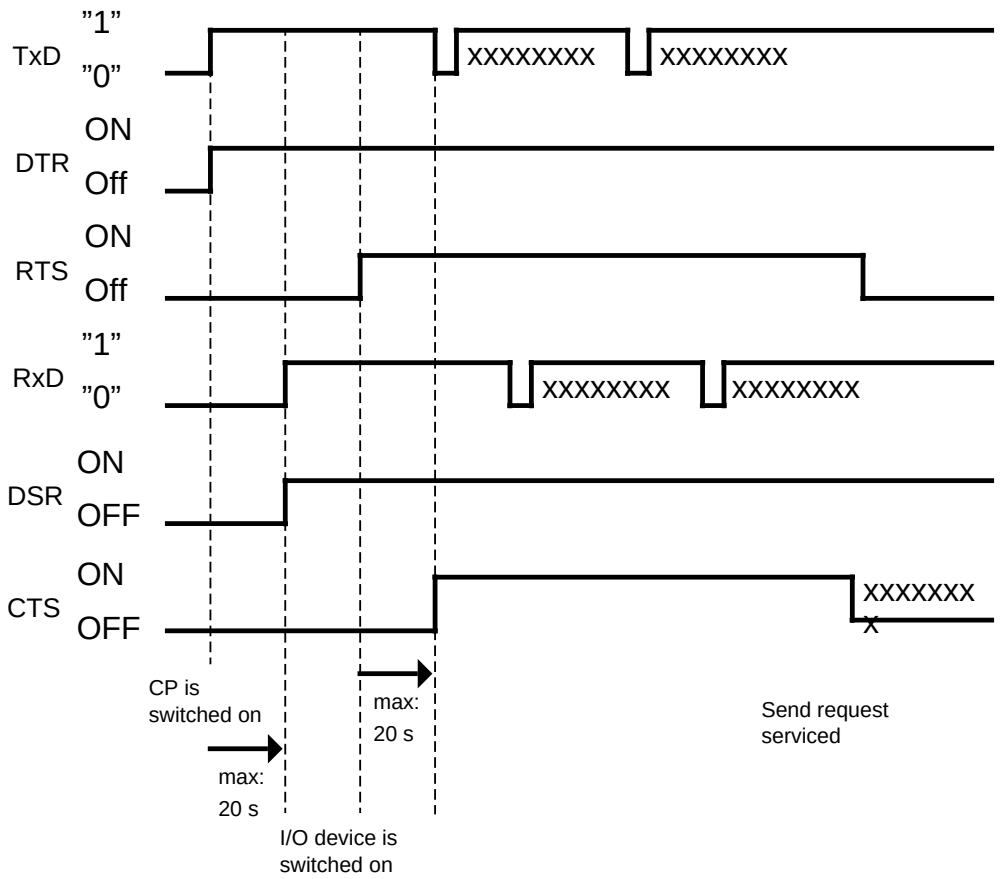


Figure 6-6. Timing Diagram for Data Transfer Between CP 521 BASIC and I/O Device

Sending data over the bidirectional serial interface

Data can be transferred from the CP 521 BASIC to an I/O device over the bidirectional interface. The I/O device must be equipped with a RS232C (V.24) or TTY interface.

Data can be transferred over the bidirectional interface with LIST, PRINT or HANDLE; HANDLE uses the same Send buffer as HANDLE#, which outputs data over the unidirectional interface.

The data to be transferred can be edited in the BASIC program before they are entered in the HANDLE Send buffer, thus making it possible to implement user-written protocols.
You will find a list of commands for implementing user protocols in Section 5.10.

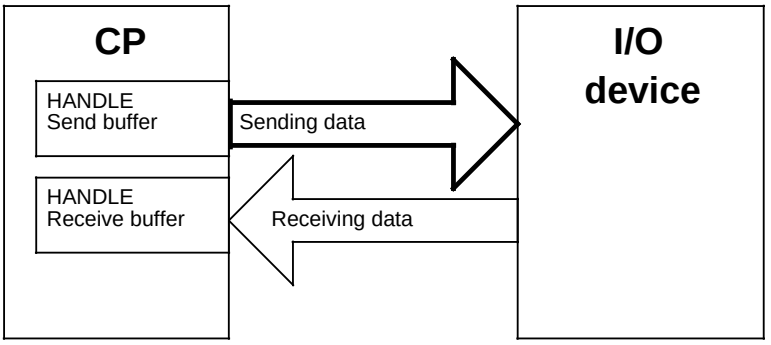


Figure 6-7. Sending Data over the Bidirectional Interface

Note:

Parallel operation of TTY and RS232C (V.24) is possible, but requires reinitialization.

Receiving data over the bidirectional serial interface

The CP 521 BASIC can receive data from an I/O device. The I/O device must be equipped with a RS232C (V.24) or TTY interface. After the data is read out of the HANDLE Receive buffer, it can be edited and postprocessed in the BASIC program.
You will find a list of commands for this purpose in Section 5.10.

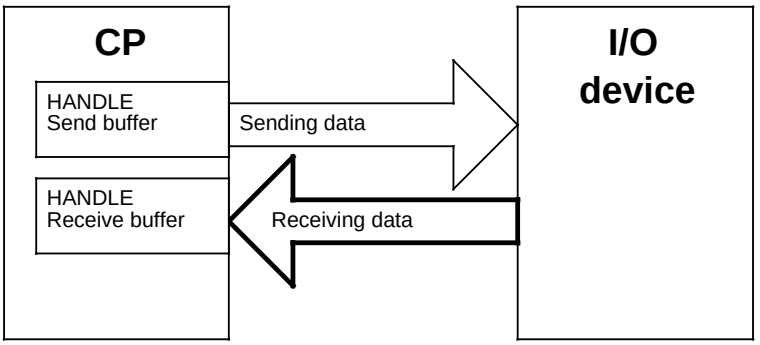


Figure 6-8. Receiving Data over the Bidirectional Interface

The data is transferred in interpretive mode.

With few exceptions, all receive characters from 00_H to FF_H are entered in the HANDLE Receive buffer, in some instances in dependence on the specified Handshake mode. The data can be read out of the HANDLE Receive buffer with GET or BUFOUT (5.10) and then postprocessed.

The XON (11_H) and XOFF (13_H) characters are not entered in the HANDLE Receive buffer. These characters are recognized and evaluated as XON/XOFF characters only when XON/XOFF protocol was specified.

The CTRL-C (03_H), CTRL-S (13_H) and CTRL-Q (11_H) characters are alway entered in the buffer and evaluated (6.4, Note). If the CTRL-C (03_H) character is to be transferred, it must be transferred with BREAK (0); otherwise the program will abort.

If XON/XOFF protocol was specified, the characters 11_H (XON, CTRL-Q) and 13_H (XOFF, CTRL-S) affect only the bidirectional interface, and they always affect PRINT and LIST.

In all other handshakes (no handshake, modem signals, PG protocol), 11_H and 13_H affect output (LIST or LIST#) over both the bidirectional and unidirectional interfaces.

6.4.2 The CP 521 BASIC's Unidirectional Serial Interface

Data can also be sent from the CP 521 BASIC to an I/O device (such as a printer) over the unidirectional serial interface. If the appropriate parameter is set, the CP 521 BASIC will evaluate the BUSY signal.

Data can be sent over the unidirectional interface with LIST#, PRINT# or HANDLE#, whereby HANDLE# uses the same HANDLE Send buffer as HANDLE (5.10.1).

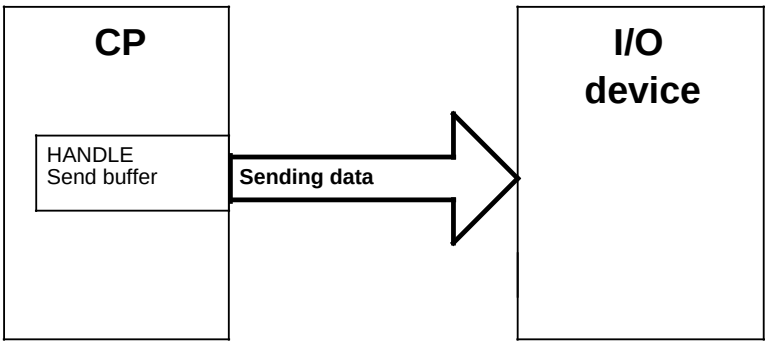


Figure 6-9. Sending Data over the Unidirectional Interface

The CP 521 BASIC's unidirectional interface can be initialized as required with INIT# (5.8.4).

7 CPU Requests and Error Messages Issued by the BASIC Interpreter

7.1 CPU Requests in BASIC Mode . . . . . 7 - 1

7.2 Error Messages Issued by the BASIC Interpreter . . . . . 7 - 2

7.2.1 Additional Error Messages . . . . . 7 - 7

Tables

7-1. Permissible Requests in BASIC Mode . . . . . 7 - 1
7-2. Error Messages Issued by the BASIC Interpreter . . . . . 7 - 3
7-3. Error Messages Issued in BASIC Mode . . . . . 7 - 7

7 CPU Requests and Error Messages Issued by the BASIC Interpreter

7.1 CPU Requests in BASIC Mode

This section lists all CPU requests which may be issued in BASIC mode.

Table 7-1. Permissible Requests in BASIC Mode

Byte 0	Byte 1	Requests
00_H		
10_H		Set CP clock
$A0_H$	01_H	Coordinate ”Send”
$A0_H$	80_H	Coordinate ”Receive”
$B0_H$	Subframe no.	Coordinate frame transfer ”CPU CP 521 BASIC”
$C0_H$	Subframe no.	Subframe acknowledgement
$D0_H$		Byte 0: Request to start a BASIC program Byte 1:
	00_H	Start the BASIC program in RAM
	01_H to $0FE_H$	Start memory submodule-resident BASIC program no. 1_H to FE_H
	FF_H	Reset the BASIC interpreter (sets the interpreter to the initial state

N O T E :

An executing BASIC program can be aborted by issuing CPU request $D100_H$. The CP 521 BASIC does not acknowledge this request.

7.2 Error Messages Issued by the BASIC Interpreter

The CP 521 BASIC provides user-friendly error detection and error handling facilities.
ONERR [ln num] (5.8.18) allows the user to program a defined reaction to arithmetic errors occurring during the program run. If no ONERR routine is programmed, the interpreter goes to STOP and displays an error message on the monitor.
If the program is aborted because of a program error or with END, STOP or CTRL-C, the interpreter exits the RUN mode, enters the Command mode, and waits for directives.
Errors can also be evaluated over the CPU. Errors can be reported to the CPU with an $A080_H$ request (”Coordinate Receive”).
The CPU is advised of error $7X_F$, which is forwarded in byte 0.
Byte 1 contains the error code. Bytes 2 to 7 contain additional information on the error location. The following table lists the error codes and the errors which they flag.

Table 7-2. Error Messages Issued by the BASIC Interpreter

Byte	Error code	Description
0	$7X_H$	Program abort due to program error or END command; the CP 521 BASIC is in PROG mode.
1	00_H	BASIC program terminated without error or aborted with STOP or CTRL-C.
	01_H	DIVIDE BY ZERO*
	02_H	ARITH. OVERFLOW*
	03_H	ARITH. UNDERFLOW*
	04_H	BAD ARGUMENT*
	05_H	BAD SYNTAX
	06_H	NO DATA
	07_H	A-STACK
	08_H	C-STACK
	09_H	I-STACK OVERFLOW
	$0B_H$	ARRAY SIZE
	$0C_H$	MEMORY ALLOCATION
	$0D_H$ to $0F_H$	Reserved for additional system messages

* Only when these errors are not intercepted with ONERR

Table 7-2. Error Messages Issued by the BASIC Interpreter (continued)

Byte	Error code	Description
2	0 1 to $0FE_H$	Contains the program number of the last program executed Program is in RAM Program is on memory submodule
3	00_H	Unassigned
4 + 5	0 to $0FF_H$	Program line number on which the error occurred (HIGH BYTE FIRST)
6	00_H	Unassigned
7	00_H	Unassigned

Once an error has been detected, the program cannot be restarted by issuing a ”$D0_H$” request to the CPU.
The errored program cannot be corrected over the CPU.

Error codes

Errors 01_H to 04_H can be intercepted with ONERR, thus making it possible to circumvent a program abort.

DIVISION BY ZERO (01_H)
A DIVIDE BY ZERO error is flagged every time an attempt is made to divide by zero, e.g. 12/0.

ARITH. OVERFLOW (02_H)
An ARITH. OVERFLOW error is flagged when the result of an arithmetic operation exceeds the upper limiting value for a BASIC floating-point number. Floating-point numbers may not exceed ±0.9999999E +127. An ARITH. OVERFLOW error would therefore be flagged for 1.0E +70*1.E +70.

ARITH. UNDERFLOW (03_H)
An ARITH. UNDERFLOW error is flagged when the result of an arithmetic operation falls below the lower limiting value for a BASIC floating-point number. The lowest floating-point number is ±1.0E - 127. The value 1.0E - 80/1.0E +80 would therefore be flagged as ARIT_H. UNDERFLOW error.

BAD ARGUMENT (04_H)
A BAD ARGUMENT error is flagged when the argument for an operator is not within the range permissible for that operator.

BAD SYNTAX (05_H)
BAD SYNTAX means that an invalid command, statement or operator was entered and cannot be processed by the BASIC interpreter. Check your entries to make sure they are correct. ***The use of a reserved keyword as part of a variable is also flagged as BAD SYNTAX error.***

NO DATA (06_H)
ERROR: No DATA - IN LINE XXX is output to the console when a READ statement was programmed without a DATA statement or when all arguments in the DATA statement have been read but no RESTORE statement was programmed.

A-STACK (07_H)
An A-STACK (argument stack) error is flagged when an attempt was made to set the A-STACK POINTER to an invalid value, which takes the pointer out of range. This can happen when you attempt to PUSH too many expressions onto the argument stack or to POP data when the stack is empty.

C-STACK (08_H)
A C-STACK (control stack) error is flagged on a C-STACK overflow. The C-STACK reserves 158 bytes in the module's RAM. Each FOR-NEXT loop requires 17 bytes, each DO-UNTIL, DO-WHILE and GOSUB loop three bytes. This means that the BASIC interpreter can process no more than nine nested FOR-NEXT loops, as 9 x 17 is 153. A C-STACK error is flagged when the programmer attempts to nest more loops than the stack can accommodate. A C-STACK error is also flagged when a RETURN precedes the associated GOSUB, a WHILE or UNTIL the associated DO, or a NEXT the associated FOR statement.

I-STACK (09_H)
An I-STACK (internal stack) error is flagged when there is not enough room left in the stack to process a numeric expression. One way of circumventing an I-STACK error is to shorten extremely long expressions in statements (perhaps by dividing them into several shorter expressions).

ARRAY SIZE ($0B_H$)
An ARRAY SIZE error is flagged when, after dimensioning an array with DIM, an attempt was made to address a variable located outside the bounds of that array (default dimension max. 10 elements).

MEMORY ALLOCATION ($0C_H$)
A MEMORY ALLOCATION error is flagged when the user attempts to address STRINGS that are ”outside” the defined string area. This error is also flagged when there is not enough memory left to allocate or declare additional variables; for example, String 250,250 or invalid value assigned to MTOP.

7.2.1 Additional Error Messages

The CP 521 BASIC reports additional errors in byte 0.
Byte 0, which provides information on the module status, is divided into a high-order and a low-order nibble. When several errors occur simultaneously, they are flagged on a priority basis. The higher the value in a nibble, the higher the priority of the error flagged by that value.

Table 7-3. Error Messages Issued in BASIC Mode

Byte 0 bits 4 to 7	bits 0 to 3	Status
X	0	No errors detected
X	7	Battery defect
X	F	Module is busy; CPU request cannot be accepted
1	X	Clock defect
2	X	Clock set to default value
3	X	Time/date error
4	X	Invalid request
5	X	Coordination info (Send - Receive)
6	X	Data transfer CP 521 BASIC CPU
7	X	BASIC error message (7.2)
8	X	Hardware error

X= Signal state irrelevant to other nibble

8 Programs

8.1 Introduction . . . . . 8 .- 1

8.2 Configuration . . . . . 8 .- 2

8.3 Interface Initialization . . . . . 8 - 2

8.4 Programming the CP 521 BASIC . . . . . 8 - 3

8.5 Programming the PLC . . . . . 8 - 7

8.5.1 Overview of Blocks . . . . . 8 - 7
8.5.2 Program Structure . . . . . 8 - 9
8.5.3 Format of the Status Byte . . . . . 8 - 10
8.5.4 Program Blocks OB 1 and FB 10 ”PULSE” . . . . . 8 - 11
8.5.5 Function Block FB 11 ”DISTRIB” . . . . . 8 - 12
8.5.6 Function Block FB 1
Send Fixed-Length Frame . . . . . 8 - 13
8.5.7 Function Block FB 2 (not for CPU 100 and CPU 102)
Send Frame with End Characters . . . . . 8 - 15
8.5.8 Function Block FB 3
Receive Fixed-Length Frame . . . . . 8 - 17
8.5.9 Programmable Function Block FB 200 ”SEND” . . . . . 8 - 19
8.5.10 Programmable Function Block FB 201 ”RECEIVE” . . . . . 8 - 27

Figures

8-1. Configuration Schematic CPU and the CP 521 BASIC for Programming . . . . . 8. - 2
8-2. BASIC Program Structure . . . . . 8 - 3
8-3. Block Diagram for ”Receive” . . . . . 8 - 4
8-4. Block Diagram for ”Send” . . . . . 8 - 5
8-5. Program Structure . . . . . 8. - 9
8-6. Format of the Status Byte (FY 80) . . . . . 8 - 10
8-7. Block Diagram for ”Send Frame” (FB 200) . . . . . 8 - 20
8-8. Block Diagram for ”Receive Frame” (FB 201) . . . . . 8 - 28

Tables

8-1. Block Parameters for FB 200 ”SEND” . . . . . 8 - 19
8-2. Overview of Flags Used by FB 200 ”SEND” . . . . . 8 - 22
8-3. Block Parameters for FB 201 ”RECEIVE” . . . . . 8 - 27
8-4. Overview of Flags Used in FB 201 ”RECEIVE” . . . . . 8 - 30

8 Programs

8.1 Introduction

This section provides detailed information on a sophisticated operator interface which greatly simplifies use of the requests ”Send frame to CP 521 BASIC” ($A001_H$) and ”Receive frame from CP 521 BASIC” ($A080_H$).
It also provides information on how to program the programmable controller (PLC) and the CP 521 BASIC.

In Sections 8.5.6 to 8.5.10 you will find **pre-programmed** function blocks for sending and receiving **fixed-length** frames and frames with **end characters** as well as a number of **programmable** function blocks, in Sections 8.5.4 and 8.5.5 ”utility” programs for problem-free, error-free data interchange between CPU and CP 521 BASIC.
In Section 8.4 you will find a BASIC program that shows you how to program the CP 521 BASIC to receive frames from the CPU and send data to the CPU.

To transfer a data frame, you need only
- enter the relevant programs over the programmer or terminal
- transfer blocks to the programmable controller and a BASIC program to the CP 521 BASIC
- enter frames in a data block and
- initialize the Send or Receive FB

You can also develop your own Send and Receive programs. You will find information for developing these programs and locating errors in the Section entitled ”Communications Interfaces” (Section 6).

8.2 Configuration

The configuration is illustrated in Figure 8.1. The CP 521 BASIC is plugged into slot 0. A programmer must be interfaced to the CPU for programming the PLC. The CP 521 BASIC and the terminal are connected to one another by a RS232C (V.24) cable. You will find sample connector pin assignments in Section 3.5.

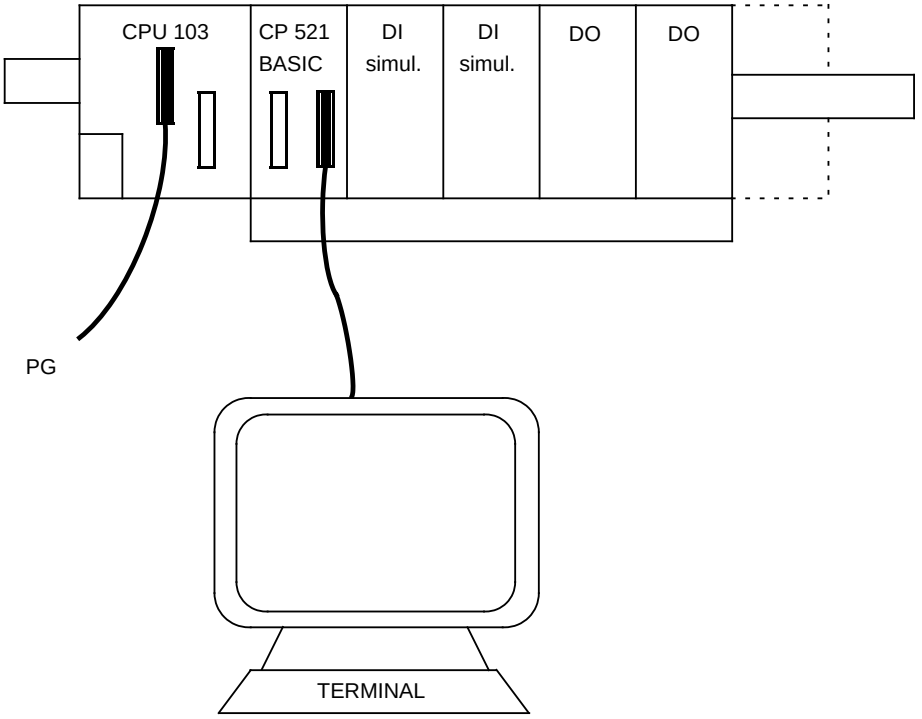


Figure 8-1. Configuration Schematic CPU and the CP 521 BASIC for Programming

8.3 Interface Initialization

Proceed as described in Section 6.2 to initialize the serial interface. When you put the CP 521 BASIC into operation for the first time, you must press the Space bar.

8.4 Programming the CP 521 BASIC

When you have switched on the CP 521 BASIC and received the SIGN ON (5.2), you can begin entering the BASIC program. The CP is in Command mode.
Use the BASIC program shown below to dimension Receive and Send variables. Display the status of the last Send or Receive request on the terminal monitor. It has been assumed that the CP 521 BASIC sends 12-byte data frames to the CPU. The CPU sends 12 bytes of data to the CP 521 BASIC.
The BASIC program shown below is only a sample program. You can modify it to meet your specific requirements.

Block diagram of the BASIC program

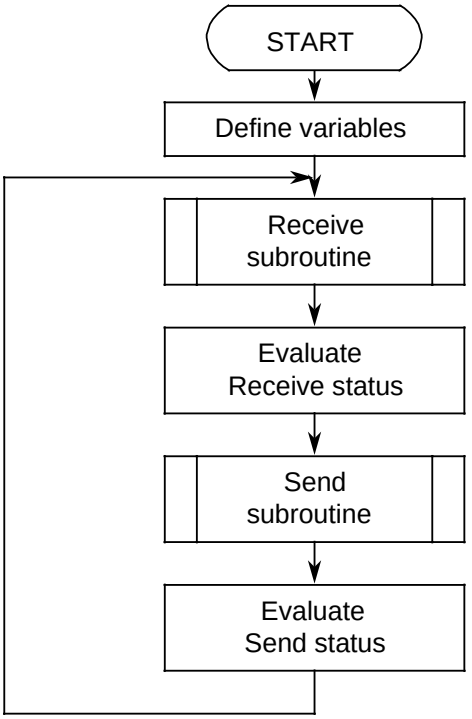


Figure 8-2. BASIC Program Structure

Block diagram of the Receive subroutine

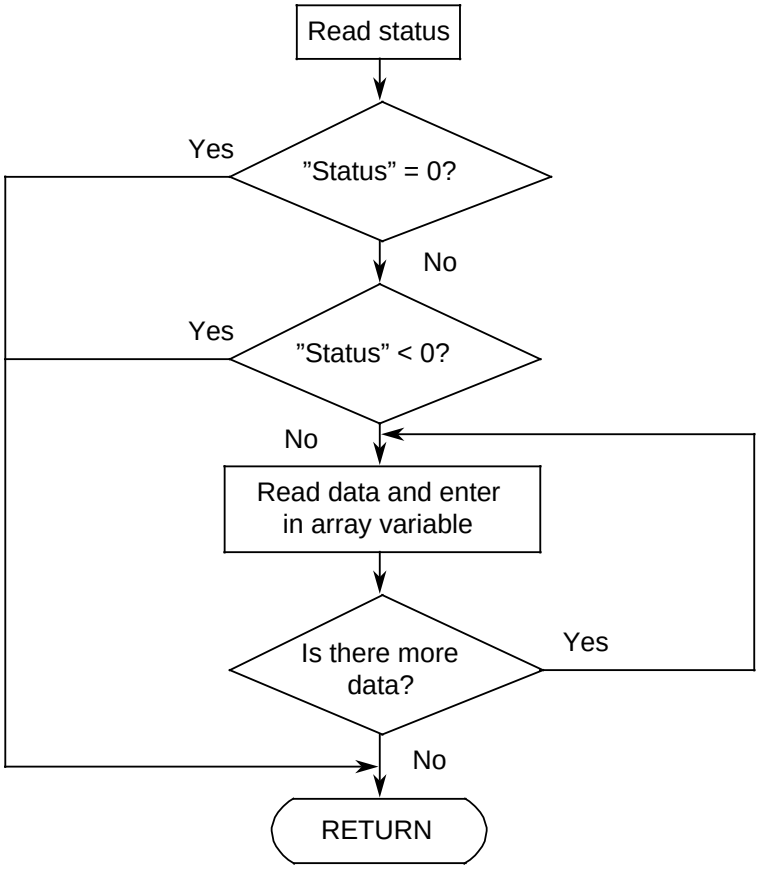


Figure 8-3. Block Diagram for ”Receive”

Note:

Data can be read in only when the CP 521 BASIC has received a frame from the CPU.

Block diagram of the Send subroutine

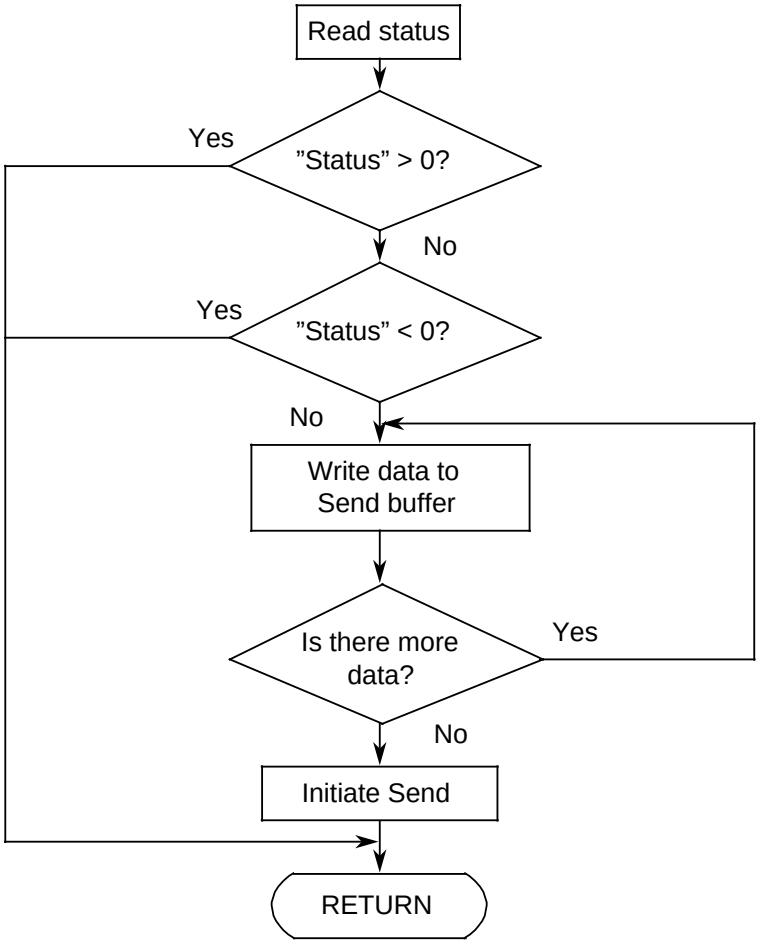


Figure 8-4. Block Diagram for ”Send”

Note:

The next Send can be initiated only when the PLC has fetched the Send data.

BASIC program - Status of the last data transfer

```
>10   DIM EVAR (254), SVAR (254)
>11   REM DIMENSION SEND AND RECEIVE VARIABLES
>20   MAX = 12 : REM NUMBER OF BYTES TO BE TRANSFERRED

>100  REM MAIN PROGRAM
>110  GOSUB 1000 : REM  RECEIVE DATA FROM PLC
>120  IF RS =   0 PRINT "NO DATA AVAILABLE"
>130  IF RS = - 1 PRINT "S5 BUS INACTIVE"
>140  IF RS = - 2 PRINT "PLC STILL SENDING DATA"
>150  IF RS = - 3 PRINT "FRAME RECEIVE ERROR"
>160  IF RS >   0 PRINT RS "BYTES RECEIVED"
>200  GOSUB 2000 : REM SEND DATA TO PLC
>210  IF SS >   0 PRINT "LAST SEND REQUEST NOT YET SERVICED"
>220  IF SS = - 1 PRINT "S5 BUS INACTIVE"
>230  IF SS = - 2 PRINT "ERROR ON SEND TO PLC (TIME-OUT)"
>240  IF SS = - 3 PRINT "ERRONEOUS ACKNOWEDGEMENT FROM CPU"
>250  IF SS =   0 PRINT MAX, "BYTES CAN BE RECEIVED FROM THE CPU"
>300  GOTO 100

>1000 REM SUBROUTINE FOR RECEIVING BYTE VARIABLES FROM THE PLC"
>1010 RS = RSTAT : IF RS = 0 GOTO 1070 : REM READ RECEIVE STATUS
>1020 IF RS < 0 GOTO 1070
>1030 FOR I = 0 TO (RS-1)       : REM  RECEIVE VARIABLES FROM 0 TO (RS-1)
>1040 BR=BUFREAD 1, EVAR (I)    : REM  READ OUT OF RECEIVE BUFFER
>1050 IF BR < > 0 THEN STOP
>1060 NEXT I
>1070 RETURN

>2000 REM SUBROUTINE FOR SENDING BYTE VARIABLES TO THE PLC
>2010 SS = SSTAT : IF SS > 0 GOTO 2080 : REM READ SEND STATUS
>2020 IF SS < 0 GOTO 2080
>2030 FOR I = 0 TO (MAX-1)      : REM  SEND VARIABLES FROM 0 TO (MAX-1)
>2040 BW = BUFWRITE 1, SVAR (I) : REM  WRITE TO SEND BUFFER
>2050 IF BW < > 0 THEN STOP
>2060 NEXT I
>2070 SEND MAX
>2080 RETURN
```

8.5 Programming the PLC

8.5.1 Overview of Blocks

Before you enter the following statement lists on the programmer and transfer them to the PLC, you should know how the control program works.
The control program is based on a cycle time of 200 ms. Every 200 ms, the PLC is alternately clear to send **or** clear to receive. Error-free interchange of frames between the CPU and the CP 521 BASIC is possible only when the last Send or Receive request has been fully serviced, i.e. Send and Receive must be mutually ”interlocked”.

The following blocks are provided for this purpose:

- **Organization block OB 1 (8.5.4):**
  Blocks invoked in OB 1 are processed cyclically.

- **Function block FB 10 ”PULSE” (8.5.4):**
  A pulse flag (F 100.0) is set at periodic intervals (every 200 ms) and reset at the beginning of the next program cycle.

- **Function block FB 11 ”DISTRIB” (8.5.5):**
  Permission to send or permission to receive is granted alternately at periodic intervals (every 200 ms).

- **Non-programmable Send and Receive FBs:**
  - **FB 1 (8.5.6)**: Send fixed-length frame
  - **FB 2 (8.5.7)**: Send frame with end character(s)
  - **FB 3 (8.5.8)**: Receive fixed-length frame

- **Programmable Send and Receive FBs:**
  - **FB 200 ”SEND”**: Send variable-length frame (8.5.9)
  - **FB 201 ”RECEIVE”**: Receive variable-length frame (8.5.10)

Note:

FB 2, FB 200 ”SEND” and FB 201 ”RECEIVE” execute only on the CPU 103.
The ”DO FLAG WORD” operation (DO FW) is not part of the CPU 100 or CPU 102 operation set.

Observe the following when entering blocks:

- OB 1, FB10 ”PULSE” and FB11 ”DISTRIB” are all-purpose blocks.

Note:

If you want to replace pre-programmed Send and Receive blocks with other pre-programmed Send and Receive blocks, or if you want to program your own, you need only correct the block numbers in the ”block call” statements in FB 11 ”DISTRIB” (Figure 8.5).

- Before invoking a Send routine, you must enter the frame to be transferred in a data block (source data block).

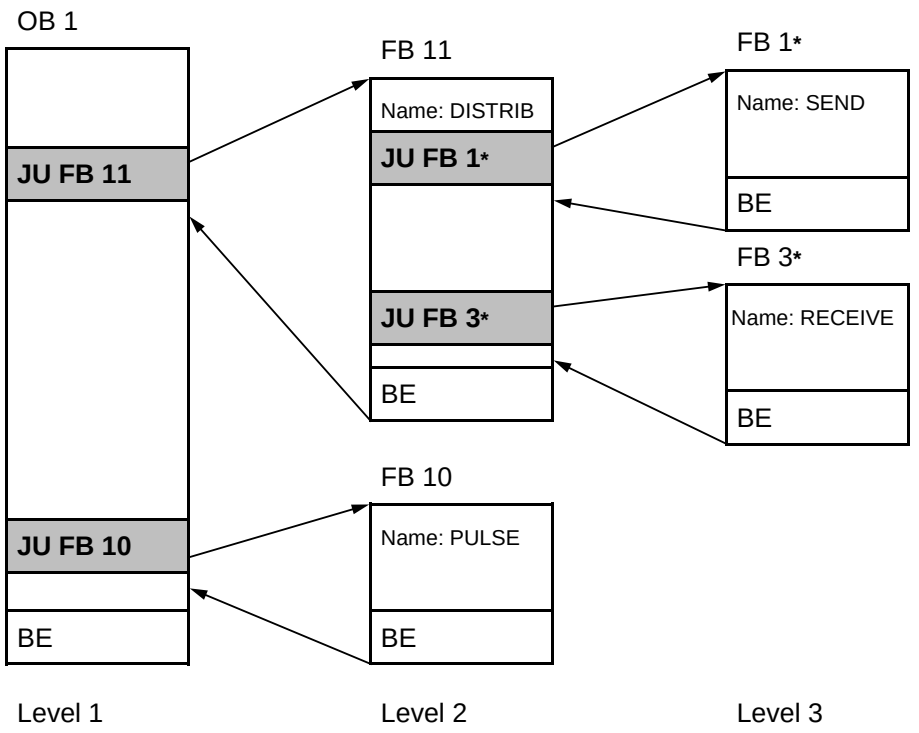
Note:

The block number of the source data block must be the same as the block number in the ”Call data block” statements.

- Flag byte 80 (FY 80) is reserved for the data transfer status in all of the following FBs.

8.5.2 Program Structure

Our control program has three nesting levels.
The function blocks invoked in OB 1 are processed cyclically.



* FB 1 and FB 3 can be replaced by other pre-programmed SEND and RECEIVE FBs (FB 2, FB 200, FB 201) or by other user-written FBs.

Figure 8-5. Program Structure

8.5.3 Format of the Status Byte

Flag byte 80 is the status byte for all Send and Receive FBs listed in Section 8.
The status byte (FY 80) provides information on

- whether a Send or Receive request was issued
  (bit 0=1 or bit 4=1, respectively)
- whether a Send or Receive request is currently being serviced
  (bit 2=1 or bit 6=1, respectively)
- whether an error occurred.
  Bit 1=1 or bit 5=1 indicates that the request was rejected (it was not possible to send or receive data).

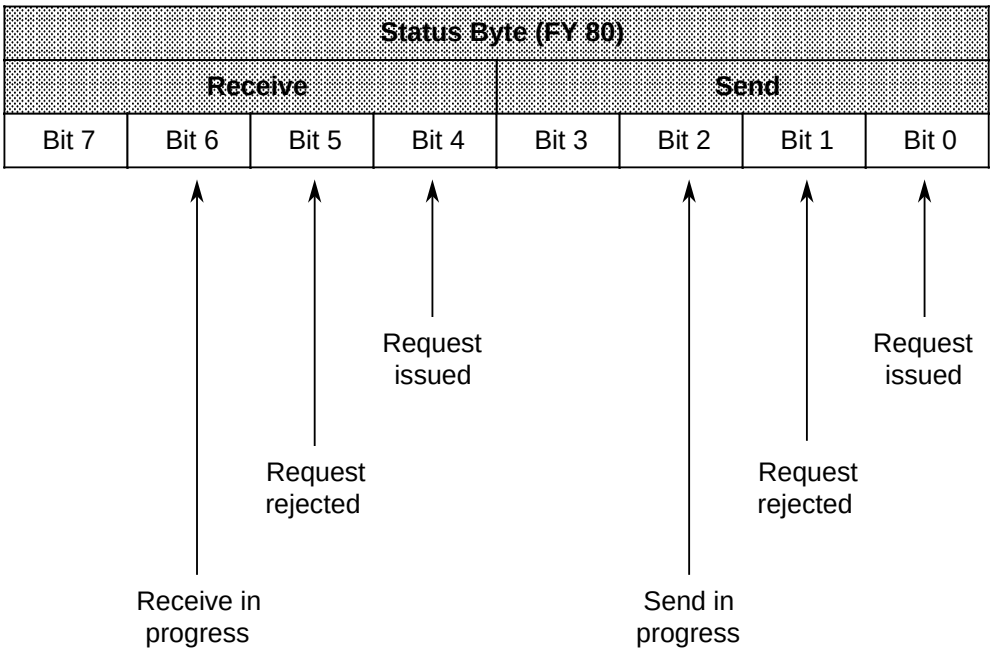


Figure 8-6. Format of the Status Byte (FY 80)

8.5.4 Program Blocks OB 1 and FB 10 ”PULSE”

STL OB 1	Description
SEGMENT 1	
:JC FB 11	Invoke auxiliary FBs:
NAME :DISTRIB	Grant permission to send or receive
:	
:JU FB 10	
NAME :PULSE	Pulse generator
:	
:BE	

STL FB 10	Description
SEGMENT 1	
NAME :PULSE	Scan and reset pulse flag 100.0 at the
:A F 100.0	beginning of each new program cycle
:R F 100.0	
:	
:	
:AN F 4.0	
:L KT 20.0	Cycle time: here 200 ms
:SI T 4	Start pulse timer
:AN T 4	
:AN F 4.0	
:= F 4.0	
:S F 100.0	Set pulse flag 100.0 after 200 ms
:B	

8.5.5 Function Block FB 11 ”DISTRIB”

STL FB 11	Description
SEGMENT 1	
NAME :DISTRIB	Pulse flag F 100.0 enables frame transfer.
:A F 100.0	
:AN F 100.1	
:S F 100.1	
:S F 80.2	Send in progress
:BEB	
:A F 100.0	
:A F 100.1	
:R F 100.1	
:S F 80.6	Receive in progress
:BEC	
:	
:AN F 80.2	Send FB is processed if permission to send has been granted.
:JC =M001	
:	
:JU FB 1*	Invoke the Send FB
NAME :SEND	
:BEU	
:	
M001 :AN F 80.6	Receive FB is processed if permission to receive has been granted.
:JC =M002	
:	
:JU FB 3*	Invoke Receive FB
NAME :RECEIVE	
:	
M002 :BE	

* FB 1 and FB 3 can be replaced by other pre-programmed FBs (FB 2, FB 200, FB 201) or by user-written FBs.

8.5.6 Function Block FB 1 Send Fixed-Length Frame

Let us assume that you have plugged the CP 521 BASIC into slot 5 (start address 104) and want to transfer a fixed-length frame (12 bytes) from the CPU to the CP 521 BASIC. You have entered the frame data in data block (DB) 21.

STL FB 1	Description
SEGMENT 1 0000 NAME :SEND	Send fixed-length frames (12 bytes)
0005 :C DB 21	Open Send FB
0006 :	
0007 :L IW 104	CP busy ?
0008 :L KH 0F00	
000A :AW	
000B :L KH 0F00	
000D :!=F	
000E :BEC	
000F :	
0010 :A F 80.0	Send request already
0011 :	issued
0012 :JC = M001	
0013 :L KH A001	Issue Send request
0015 :T QW 104	
0016 :	
0017 :L KH 000C	Forward Send length
0019 :T QW 106	(Send length + 12 bytes)
001A :	
001B :L KH 0000	Forward end char.
001D :T QW 108	(Since Send length is > 0,
001E :	end char. is irrelevant)
001F :S F 80.0	
0020 :R F 80.1	
0021 :BEU	
0022 :	
0023 M001 :L IW 104	Evaluate acknowledgement
0024 :L KH F00F	
0026 :AW	
0027 :L KH 5001	
0029 :><F	
002A :JC =M002	
002B :	
002C :L IW 106	Request OK ?
002D :L KH 0000	
002F :><F	
0030 :JC =M003	
0031 :	
0032 :L KH B001	Transfer frame 1

STL FB 1 (continued)	Description

```
0034      :T   QW 104
0035      :
0036      :L   DW   0      Forward data word 0
0037      :T   QW 106
0038      :L   DW   1      Forward data word 1
0039      :T   QW 108
003A      :L   DW   2      Forward data word 2
003B      :T   QW 110
003C      :BEU
003D      :
003E M003 :L   IW 106      Eval. acknowl. for frame 1
003F      :L   KH 0001
0041      :AW
0042      :L   KH 0001
0044      :><F
0045      :JC =M002
0046      :
0047      :L   KH B002    Transfer frame 2
0049      :T   QW 104
004A      :L   DW   3     Forward data word 3
004B      :T   QW 106
004C      :L   DW   4     Forward data word 4
004D      :T   QW 108
004E      :L   DW   5     Forward data word 5
004F      :T   QW 110
0050      :BEU
0051      :
0052 M002 :L   IW 104     Evaluate final acknowl.
0053      :L   KH F00F
0055      :AW
0056      :L   KH 5000
0058      :><F
0059      :JC =M004
005A      :
005B      :AN  I  106.0   No errors ?
005C      :JC =M005
005D      :
005E      :AN  F   80.1   On error, set
005F      :S   F   80.1   error flag
0060 M005 :R   F   80.2
0061      :R   F   80.0
0062      :
0063      :L   KH 0000    Clear final acknowl.
0065      :T   QW 104
0066 M004 :BE
```

8.5.7 Function Block FB 2 (not for CPU 100 and CPU 102) Send Frame with End Characters

You have plugged the CP 521 BASIC into slot 5 (start address 104) and want to send a variable-length frame from the CPU to the CP 521 BASIC. The frame data is taken from DB 22. The two end characters identify the last word.

STL FB 2	Description
SEGMENT 1 0000	Send with end char.
NAME :SEND	
0005 :C DB 22	Open Send DB
0006 :	
0007 :L IW 104	CP busy ?
0008 :L KH 0F00	
000A :AW	
000B :L KH 0F00	
000D :!=F	
000E :BEC	
000F :	
0010 :A F 80.0	Send request already
0011 :	issued
0012 :JC =M001	
0013 :L KH A001	Issue Send request
0015 :T QW 104	
0016 :	
0017 :L KH 0000	Forward Send length
0019 :T QW 106	Here: 0000H to force
001A :	interpretation of end char.
001B :L KH 000D	Forward end char.
001D :T QW 108	Here: 000DH
001E :	
001F :S F 80.0	
0020 :R F 80.1	
0021 :BEU	
0022 :	
0023 M001 :L IW 104	Evaluate acknowledgement
0024 :L KH F00F	
0026 :AW	
0027 :L KH 5001	
0029 :><F	
002A :JC =M002	
002B :	
002C :L IW 106	Request OK?
002D :L KH 0000	
002F :><F	

STL FB 2 (continued)	Description

```
0030      :JC =M003
0031      :
0032      :L   KH B001     Forward frame 1
0034      :T   QW 104
0035      :
0036      :L   DW   0      Forward data word 0
0037      :T   QW 106
0038      :L   DW   1      Forward data word 1
0039      :T   QW 108
003A      :L   DW   2      Forward data word 2
003B      :T   QW 110
003C      :BEU
003D      :
003E M003 :L   IW 106      Eval. acknowl. for frame 1
003F      :L   KH 0001
0041      :AW
0042      :L   KH 0001
0044      :><F
0045      :JC =M002
0046      :
0047      :L   KH B002     Transfer frame 2
0049      :T   QW 104
004A      :L   DW   3      Forward data word 3
004B      :T   QW 106
004C      :L   DW   4      Forward data word 4
004D      :T   QW 108
004E      :L   DW   5      Forward data word 5
004F      :T   QW 110
0050      :BEU
0051      :
0052 M002 :L   IW 104      Eval. final acknowl.
0053      :L   KH F00F
0055      :AW
0056      :L   KH 5000
0058      :><F
0059      :JC =M004
005A      :
005B      :AN  I  106.0    No errors ?
005C      :JC =M005
005D      :
005E      :AN  F   80.1    On error, set
005F      :S   F   80.1    error flag
0060 M005 :R   F   80.2
0061      :R   F   80.0
0062      :
0063      :L   KH 0000     Clear final acknowl.
0065      :T   QW 104
0066 M004 :BE
```

8.5.8 Function Block FB 3 Receive Fixed-Length Frame

You have plugged the module into slot 5 (start address 104) and want to transfer 12-byte data frames from the CP 521 BASIC to the CPU. You are using data block (DB) 23 to store the incoming data.

STL FB 3	Description
SEGMENT 1 0000 NAME :RECEIVE	Receiving max. 12 bytes (incl. end char.);
0005 :C DB 23	Open Receive DB
0006 :	
0007 :L IW 104	CP busy ?
0008 :L KH 0F00	
000A :AW	
000B :L KH 0F00	
000D :!=F	
000E :BEC	
000F :	
0010 :A F 80.4	Receive request already
0011 :JC=M001	issued
0012 :	
0013 :L KH A080	Forward Receive request
0015 :T QW 104	
0016 :S F 80.4	
0017 :R F 80.5	Reset error flag
0018 :BEU	
0019 :	
001A M001 :L IW 104	Eval. acknowl. for frame 1
001B :L KH F00F	
001D :AW	
001E :L KH 6001	
0020 :><F	
0021 :JC =M002	
0022 :T FW 90	
0023 :	
0024 :L IW 106	Read data word 0
0024 :T DW 0	
0026 :L IW 108	Read data word 1
0027 :T DW 1	
0028 :L IW 110	Read data word 2
0029 :T DW 2	
002A :	
002B :L KH C001	AcknoWledge frame 1
002D :T QW 104	
002E :BEU	
002F :	

STL FB 3 (continued)	Description
0030 M002 :L IW 104	Eval. acknowl. for frame 2
0031 :L KH F00F	
0033 :AW	
0034 :L KH 6002	
0037 :><F	
0038 :JC =M003	
0039 :T FW 90	
003A :	
003B :L IW 106	Read data word 3
003C :T DW 3	
003D :L IW 108	Read data word 4
003E :T DW 4	
003F :L IW 110	Read data word 5
0040 :T DW 5	
0041 :	
0042 :L KH C002	Acknowl. frame 2
0044 :T QW 104	
0045 :BEU	
0046 :	
0047 M003 :L IW 104	Eval. final acknowl.
0048 :L KH F00F	
004A :AW	
004B :L KH 5000	
004D :><F	
004E :JC =M004	
004F :	
0050 :AN I 106.0	No errors ?
0051 :JC =M005	
0052 :	
0053 :AN F 80.5	
0054 :S F 80.5	
0055 M005 :R F 80.6	
0056 :R F 80.4	
0057 :L IB 109	
0058 :T FY 82	
0059 :L KH 0000	Clear final acknowl.
005A :T QW 104	
005B M004 :BE	

8.5.9 Programmable Function Block FB 200 ”SEND”

FB 200 (”SEND”) transfers a variable-length frame from the CPU to the CP 521 BASIC. Before invoking FB 200 ”SEND”, you must enter the frame to be transferred in a data block (source DB). The following information must be specified in the FB 200 call:

- Start address of the CP 521 BASIC
- Number of the source data block from which the frame is to be transferred to the CP 521 BASIC
- Number of the source data word at which the frame begins
- Length of the frame to be transferred (number of source data bytes)

Note:

The FB 200 ”SEND” does neither execute on the CPU 100 nor the CPU 102.

Calling and initializing FB 200 ”SEND”*

STL	LAD/CSF

`:JU FB 200`
`NAME :SEND`
`ID :BGAD :KF`
`ID :Q-DB :B`
`ID :QANF :KF`
`ID :QLAE :KF`
	FB 200
	BGAD
	Q-DB
	QANF
	QLAE

Table 8-1. Block Parameters for FB 200 ”SEND”

Name (identifier)	Param. type	Data type	Description
BGAD	D	KF	Module start address
Q-DB	B		Number of the source data block
QANF	D	KF	Number of the first source data word
QLAE	D	KF	Number of data bytes to be transferred

* Please note that scratch flags are used in FB 200.

Block diagram of FB 200 ”SEND”

Note:

The information presented in this section has been included only to show how FB 200 ”SEND” works, and is not needed to initialize or use the block.

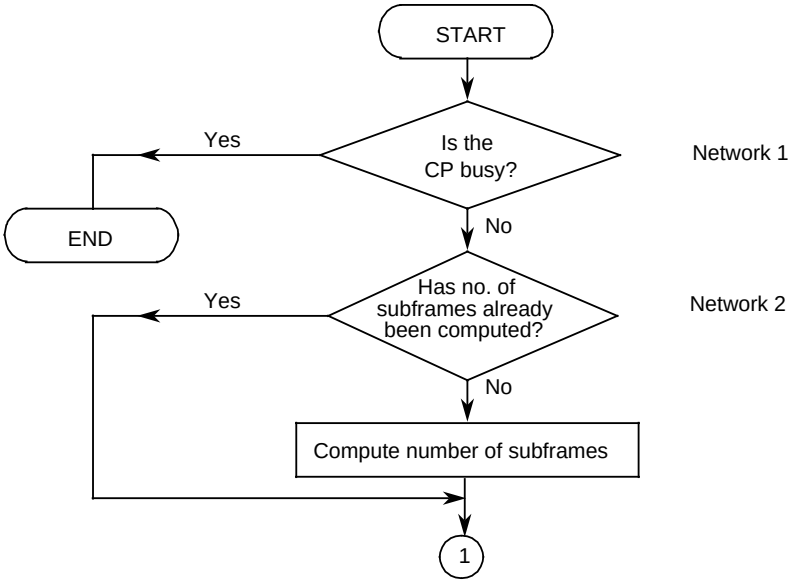


Figure 8-7. Block Diagram for ”Send Frame” (FB 200)

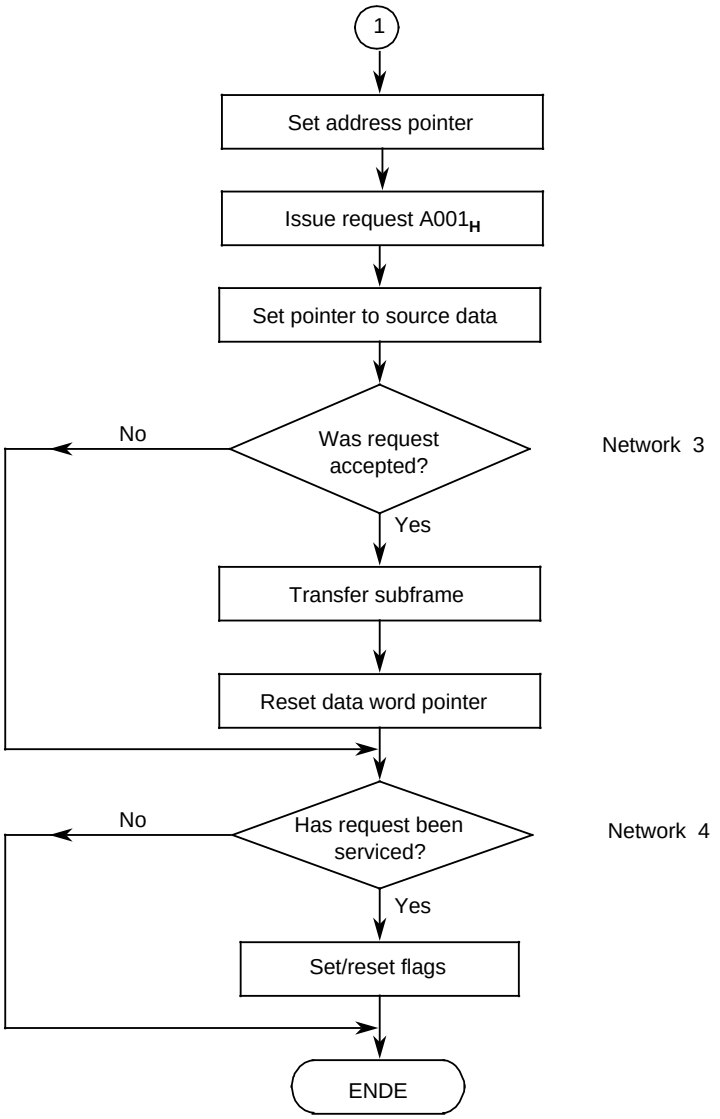


Figure 8-7. Block Diagram for "Send Frame" (continued)

Flags used in FB 200 ”SEND”

List of the flags used by FB 200 ”SEND”:

Table 8-2. Overview of Flags Used by FB 200 ”SEND”

Flag	Description
FW 234	”Word 0” of the module address register
FW 236	”Word 1” of the module address register
FW 238	”Word 2” of the module address register
FW 240	”Word 3” of the module address register
FW 242	Bit pattern for subframe announcement (which subframe?)
FW 244	Comparison pattern for subframe number
FW 246	”Word 0” of the data word pointer
FW 248	”Word 1” of the data word pointer
FW 250	”Word 2” of the data word pointer
FW 252	Auxiliary register for no. of subframes
FW 254	Auxiliary register for length of source DB
F 80.0	Flag: Request issued
F 80.1	Flag: Request rejected
F 80.2	Flag: Send in progress

STL FB 200	Description

```
SEGMENT 1     0000        Send variable-length frames
NAME :SEN  VAR
ID  :BGAD    I/Q/D/B/T/C: D  KM/KH/KY/KS/KF/KT/KC/KG: KF
ID  :Q-DB    I/Q/D/B/T/C: B
ID  :QANF    I/Q/D/B/T/C: D  KM/KH/KY/KS/KF/KT/KC/KG: KF
ID  :QLAE    I/Q/D/B/T/C: D  KM/KH/KY/KS/KF/KT/KC/KG: KF

0011      :LW  =BGAD       Load module address
0012      :T   FW 234      and store
0013      :
0014      :DO   FW 234      CP busy ?
0015      :L   IW   0
0016      :L   KH 0F00
0018      :AW
0019      :L   KH 0F00
001B      :!=F             If yes, go to end of block
001C      :BEC
001D      :***

SEGMENT 2     001E
001E      :A   F   80.0    Flag for request already issued
001F      :JC =M001        If flag is set, do not
0020      :                issue any new requests
0021      :
0022      :DO  FW 234      Check to see if request
0023      :L   IW   0      can be issued
0024      :L   KH C000
0026      :AW
0027      :L   KH 0000
0029      :><F
002A      :BEC
002B      :
002C      :L   KF +0       Reset frame counter
002E      :T   FW 252
002F      :LW  =QLAE
0030      :T   FW 254
0031 M003 :L   FW 252      Increment frame counter
0032      :ADD KF +1
0034      :T   FW 252
0035      :L   FW 254      Subtract no. of bytes in frame
0036      :ADD KF -6       from length until counter is 0
0038      :T   FW 254
0039      :L   KF +0
003B      :<=F
003C      :JC  =M002       No. of frames is reached
003D      :JU  =M003       Final no. of frames not
003E      :                yet reached
```

STL FB 200 (continued)	Description

```
003F M002 :
0040      :L   FW 234      Set address pointer
0041      :L   KF +2
0043      :+F
0044      :T   FW 236      Module address + 2
0045      :L   KF +2
0047      :+F
0048      :T   FW 238      Module address + 4
0049      :L   KF +2
004B      :+F
004C      :T   FW 240      Module address + 6
004D      :
004E      :L   KH A001     Initiate Send
0050      :DO  FW 234
0051      :T   QW   0
0052      :LW  =QLAE       Enter length in word 1
0053      :DO  FW 236
0054      :T   QW   0
0055      :L   KH B001     ID for frame acknowl.
0057      :T   FW 242      first frame - store
0058      :L   KF +0       Comparison value -
005A      :T   FW 244      frame 1 - store
005B      :
005C      :LW  =QANF       Start addr. of
005D      :T   FW 246      data block
005E      :L   KF +1
0060      :+F
0061      :T   FW 248      Start + 1
0062      :L   KF +1
0064      :+F
0065      :T   FW 250      Start + 2
0066      :
0067      :AN  F   80.0    Set flag for Send request
0068      :S   F   80.0    issued
0069      :R   F   80.1    Reset error flag
006A      :BEU
006B M001 :***

SEGMENT 3     006C
006C      :DO  FW 234      Load input word 0
006D      :L   IW   0      and check to see if
006E      :L   KH F00F     request was accepted
0070      :AW
0071      :L   KH 5001
0073      :><F
0074      :JC  =M001
0075      :
0076      :DO  FW 236      Check to see if the
0077      :L   IW   0      correct frame can be
0078      :L   KH 00FF     received
```

STL FB 200 (continued)	Description

```
007A      :AW
007B      :L   FW 244
007C      :><F
007D      :BEC
007E      :
007F      :L   FW 242      Check to see if
0080      :L   KH 00FF     next frame can be sent
0082      :AW
0083      :L   FW 252
0084      :>F
0085      :BEC
0086      :
0087      :DO  =Q-DB       Open source DB
0088      :L   FW 242      Announce and
0089      :DO  FW 234      transfer frame
008A      :T   QW   0
008C      :DO  FW 246      Word 0
008D      :L   DW   0
008E      :DO  FW 236
008F      :T   QW   0
0091      :DO  FW 248      Word 1
0092      :L   DW   0
0093      :DO  FW 238
0094      :T   QW   0
0096      :DO  FW 250      Word 2
0097      :L   DW   0
0098      :DO  FW 240
0099      :T   QW   0
009A      :
009B      :L   FW 242      Update frame acknowl.
009C      :L   KF +1
009E      :+F
009F      :T   FW 242
00A1      :L   FW 244      Update frame number
00A2      :L   KF +1
00A4      :+F
00A5      :T   FW 244
00A6      :                Increment data word pointer
00A7      :L   FW 246      Word 0
00A8      :L   KF +3
00AA      :+F
00AB      :T   FW 246
00AD      :L   FW 248      Word 1
00AE      :L   KF +3
0OB0      :+F
0OB1      :T   FW 248
0OB3      :L   FW 250      Word 2
0OB4      :L   KF +3
00B6      :+F
00B7      :T   FW 250
00B8      :BEU
00B9 M001 :***
```

STL FB200 (continued)	Description

```
SEGMENT 4      00BA
00BA      :DO  FW 234      Check for final acknowl.
00BB      :L   IW   0
00BC      :L   KH F00F
00BE      :AW
00BF      :L   KH 5000
00C1      :><F
00C2      :JC  =M001
00C3      :
00C4      :DO  FW 236
00C5      :L   IW   0
00C6      :L   KH 0100
00C8      :AW
00C9      :JZ =M002
00CA      :AN  F   80.1    Job terminated
00CB      :S   F   80.1    with error
00CC M002 :R   F   80.2    Enable for new Send or
00CD      :R   F   80.0    Receive request
00CE      :
00CF      :L   KH 0000     Clear final acknowl.
00D1      :DO  FW 234
00D2      :T   QW   0
00D3 M001 :BE
```

8.5.10 Programmable Function Block FB 201 ”RECEIVE”

FB 201 ”RECEIVE” transfers frames from the CP 521 BASIC to the CPU.
You must include the following information in the FB 201 ”RECEIVE” call:

- Start address of the CP 521 BASIC
- Number of the destination data block in which the frame from the CP 521 BASIC is to be stored
- Number of the first destination data word

Note:

FB 201 ”RECEIVE” does neither execute on the CPU 100 nor the CPU 102.

Calling and initializing FB 201 ”RECEIVE”*

STL	LAD/CSF

`:JU FB 201`
`NAME :RECEIVE`
`ID :BGAD :KF`
`ID :Z-DB :B`
`ID :ZANF :KF`
	FB 201
	BGAD
	Z-DB
	ZANF

Table 8-3. Block Parameters for FB 201 ”RECEIVE”

Name (identifier)	Param. type	Data type	Description
BGAD	D	KF	Start address of the CP 521 BASIC
Z-DB	B		Number of the destination data block
ZANF	D	KF	Number of the first destination data word

* Please note that scratch flags are used in FB 201.

Block diagram of FB 201 ”RECEIVE”

Note:

The information presented in this section has been included only to show how FB 201 ”RECEIVE” works, and is not needed to use the block.

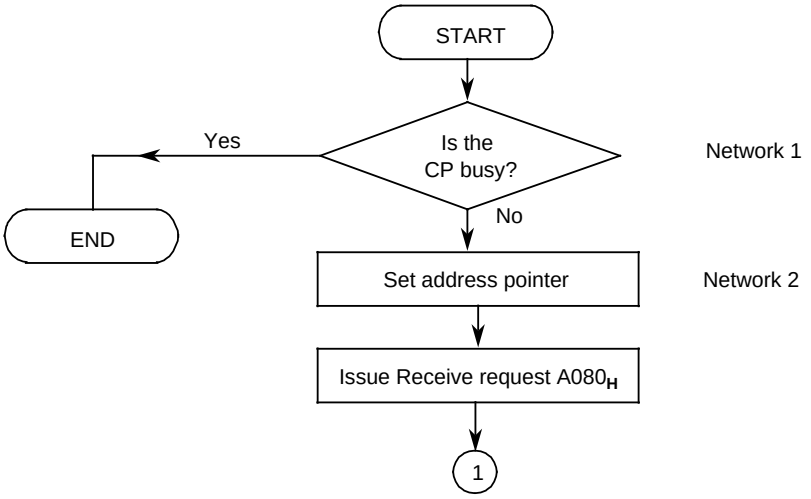


Figure 8-8. Block Diagram for ”Receive Frame” (FB 201)

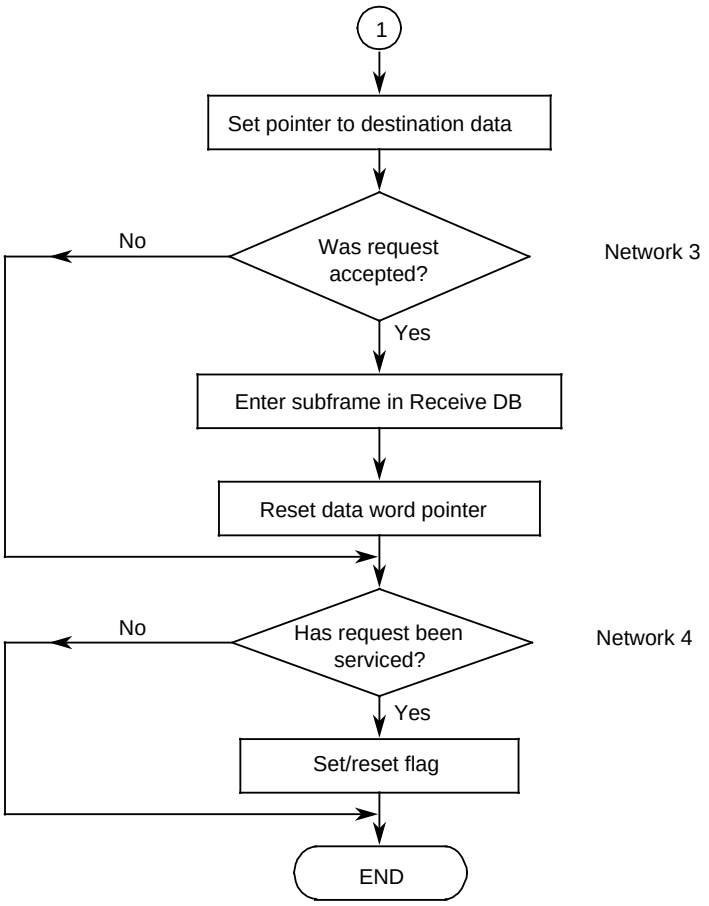


Figure 8-8. Block Diagram for ”Receive Frame” (continued)

Flags used in FB 201 ”RECEIVE”

List of the flags used in FB 201 ”RECEIVE”:

Table 8-4. Overview of Flags Used in FB 201 ”RECEIVE”

Flag	Description
FW 234	”Word 0” of the module address register
FW 236	”Word 1” of the module address register
FW 238	”Word 2” of the module address register
FW 240	”Word 3” of the module address register
FW 242	Comparison pattern for the CP subframe acknowledgement
FW 244	Subframe acknowledgement from the CPU
FW 246	”Word 0” of the data word pointer
FW 248	”Word 1” of the data word pointer
FW 250	”Word 2” of the data word pointer
F 80.4	Flag: Request issued
F 80.5	Flag: Request rejected
F 80.6	Flag: Receive in progress

STL FB 201	Description

```
SEGMENT 1    0000        Receive
NAME :EMPF VAR
ID  :BGAD   I/Q/D/B/T/C: D  KM/KH/KY/KC/KF/KT/KC/KG: KF
ID  :Z-DB   I/Q/D/B/T/C: B
ID  :ZANF   I/Q/D/B/T/C: D  KM/KH/KY/KC/KF/KT/KC/KG: KF

000E      :LW  =BGAD       Load module address
000F      :T   FW 234      and store
0010      :
0011      :DO  FW 234      CP busy ?
0012      :L   IW   0
0013      :L   KH 0F00
0015      :AW
0016      :L   KH 0F00
0018      :!=F
0019      :BEC
001A      :***


SEGMENT 2      001B
001B      :A   F   80.4    Flag for request already
001C      :JC  =M001       issued
001D      :
001E      :L   FW 234      Set address pointer
001F      :L   KF +2
0021      :+F
0022      :T   FW 236      Base address + 2 (Word 1)
0023      :L   KF +2
0025      :+F
0026      :T   FW 238      Base address + 4 (Word 2)
0027      :L   KF +2
0029      :+F
002A      :T   FW 240      Base address + 6 (Word 3)
002B      :
002C      :L   KH A080     Initiate Receive request
002E      :DO  FW 234
002F      :T   QW   0
0030      :L   KH 6001     Transfer acknowl. to
0032      :T   FW 242      flag words
0033      :L   KH C001
0035      :T   FW 244
0036      :LW  =ZANF       Set pointer to dest. data
0037      :T   FW 246      Start of dest.
0038      :L   KF +1
003A      :+F
003B      :T   FW 248      Start + 1
003C      :L   KF +1
003E      :+F
003F      :T   FW 250      Start + 2
0040      :
0041      :S   F   80.4    Set flag for Receive
0042      :BEU             initiated and go to block end
0043 M001 :***
```

STL FB 201 (continued)	Description

```
SEGMENT 3      0044
0044      :DO  FW 234      Load input word 0 and
0045      :L   IW   0      check for Receive frame
0046      :L   KH F0FF
0048      :AW
0049      :L   FW 242
004A      :><F
004B      :JC  =M001
004C      :DO  =Z-DB       Open destination DB
004D      :DO  FW 236
004E      :L   IW   0
004F      :DO  FW 246      Word for
0050      :T   DW   0      data word pointer 0
0051      :
0052      :DO  FW 238
0053      :L   IW   0
0054      :DO  FW 248      Word for
0055      :T   DW   0      data word pointer 1
0056      :
0057      :DO  FW 240
0058      :L   IW   0
0059      :DO  FW 250      Word for
005A      :T   DW   0      data word pointer 2
005B      :
005C      :L   FW 244      Acknowledge Receive data
005D      :DO  FW 234
005E      :T   QW   0
005F      :
0060      :L   FW 242      Update frame acknowl. and
0061      :L   KF +1
0063      :+F
0064      :T   FW 242
0065      :
0066      :L   FW 244      frame number
0067      :L   KF +1
0069      :+F
006A      :T   FW 244
006B      :
006C      :L   FW 246      Increment data word pointer
006D      :L   KF +3       word 0
006F      :+F
0070      :T   FW 246
0071      :
0072      :L   FW 248      Increment data word pointer
0073      :L   KF +3       word 1
0075      :+F
0076      :T   FW 248
0077      :
0078      :L   FW 250      Increment data word pointer
0079      :L   KF +3       word 2
007B      :+F
007C      :T   FW 250
007D      :BEU
007E M001 :***
```

STL FB 201 (continued)	Description

```
SEGMENT 4      007F
007F      :DO  FW 234      Check for final acknowl.
0080      :L   IW   0
0081      :L   KH F00F
0083      :AW
0084      :L   KH 5000
0086      :><F
0087      :JC  =M001
0088      :DO  FW 236
0089      :L   IW   0
008A      :L   KH 0100
008C      :AW
008D      :JZ  =M002
008E      :AN  F   80.5    Job terminated with
008F      :S   F   80.5    error
0090 M002 :R   F   80.6    Enable for new Send or
0091      :R   F   80.4    Receive request
0092      :
0093      :L   KH 0000     Clear final acknowl.
0095      :DO  FW 234
0096      :T   QW   0
0097 M001 :BE
```

Appendix A: Commands, Statements and Functions

A.1 Lists of All Commands, Statements, Functions and Operators . . . . . A .- 1

A.1 Lists of All Commands, Statements, Functions and Operators

Command, Function, Statement	Description	Example
ASC (ASCII char.) (5.12.6)	Manipulate individual characters in strings, convert a character into ASCII code	PRINT ASC(A)
BREAK ([integer]) (5.8.5)	Enable/disable program interruptability via <CTRL C>	BREAK (1)
BUFIN [strvar(,expr)] (5.10.3)	Enters the contents of the variable to the I/O Send buffer, returns an error status where applicable	X=BUFIN $(0), 250
BUFOUT [strvar(,expr)] (5.10.6)	Reads characters out of the I/O Receive buffer into the variable, returns an error status where applicable	X=BUFOUT S(0), 6
BUFREAD datatype, var {, length} (5.9.5)	Reads the specified data type from the CPU Receive buffer into the variable, returns a read or conversion error where applicable	X=BUFREAD 4, $(0), 10
BUFWRITE datatype, var {, length} (5.9.6)	Converts the variable into the specified data type and transfers the value to the CPU Send buffer; returns a read or conversion error where applicable	X=BUFWRITE 1, name
CHR (expr) (5.12.6)	Manipulates individual characters in strings, converts a number into the equivalent character	X=CHR(35H)
CLEAR (5.8.1)	Clears all internal buffers, resets all numeric variables and stacks	CLEAR
CLEARI (5.8.2)	Resets all interrupts	CLEARI
CLEARS (5.8.2)	Initializes all stacks	CLEARS
CLOCK0 (5.8.20)	Disables the timer	CLOCK0
CLOCK1 (5.8.20)	Enables the timer	CLOCK1
CONT (5.6.2)	Continues a program halted with STOP or <CTRL-C>	CONT
DATA [expr] {, [expr], ...} (5.8.6)	Specifies expressions and data to be read in with READ	DATA 50, 70, -5, PI/2

Command, Function, Statement	Description	Example
DIM [var(integer)] (5.8.7)	Reserves memory space for arrays	DIM A (20)
DO-UNTIL [rel expr] (5.8.8)	Statements between DO and UNTIL are executed until the expression following the word UNTIL is TRUE	DO A=A+1 UNTIL A=4
DO-WHILE [rel expr] (5.8.9)	Statements between DO and WHILE are executed until the expression following the word WHILE is TRUE	DO A=A+1 WHILE A<4
END (5.8.10)	Terminates the program run	END
ERASE [integer] (5.7.5)	Erases the specified program on the memory submodule	ERASE 15
FIND strvar1, strvar2 (5.12.4)	Returns the position of the first occurrence of strvar2 in strvar1 as function value	POS=FIND $(0), ”Example”
FOR-TO-{STEP}-NEXT (5.8.11)	Is used to execute a sequence of commands, statements and/or functions a specific number of times (loop)	FOR A=B TO C STEP D NEXT A
FREE (5.14.2)	Queries the amount of memory still available for variables	X=FREE
GET (5.10.5)	Reads a character from the I/O Receive buffer	A=GET
GOSUB [ln num] (5.8.12)	Jumps to the specified [ln num] in a subroutine	GOSUB 1000
GOTO [ln num] (5.8.13)	Jumps to a specific program line	GOTO 150
HANDLE [expr] (5.10.1)	Outputs ”expr” characters from the I/O Send buffer over the bidirectional interface	HANDLE 10
HANDLE# [expr] (5.10.1)	Outputs ”expr” characters from the I/O Send buffer over the unidirectional interface	HANDLE# -1
HRSTAT (5.10.4)	Returns the status of the I/O Receive buffer or a Receive error	EMP=HRSTAT
HSSTAT (5.10.2)	Returns the status of the last Send request	SEN=HSSTAT

Command, Function, Statement	Description	Example
IF-THEN-{ELSE} (5.8.15)	Used to branch to specific program sections depending on whether or not the result of a numeric expression is true	IF A=100 THEN A=0 ELSE A=A+1
INIT (#) [integer] (5.8.3, 5.8.4)	Initializes the bidirectional or unidirectional interface to the specified values	INIT 61241 INIT# 7001
INPUT [var] (5.8.16)	Used to input a string or variable	INPUT A
LET [var]=[expr] (5.8.17)	Assigns the value of an arbitrary expression to a variable or string	LET A=50
LEN (5.14.2)	Queries the length of the RAM-resident program	PRINT LEN
LIST {ln num{- ln num}} (5.6.3)	Displays a program on the monitor	LIST
LIST # {ln num{- ln num}} (5.6.4)	Outputs a program over the unidirectional interface	LIST # 50-100
MID (strvar, expr1,expr2) (5.12.3)	Assigns the substring which begins at ”strvar1” and comprises ”strvar2” characters to a string variable	$(0)=MID ($(1), 1,5)
MTOP (5.14.2)	Queries or specifies the highest address in memory	PRINT MTOP
NEW (5.6.5)	Clears all buffers and the RAM-resident BASIC program	NEW
ONERR [ln num] (5.8.18)	The program branches to the specified line number when the interpreter flags an error	ONERR 250
ONTIME [expr], [ln num] (5.8.19)	Generates an interrupt when TIME is equal to or greater than the ”expr” argument and branches to the interrupt routine which begins at the specified line number	ONTIME 10, 1000
ON [expr] GOSUB [ln num] {, [ln num],...[ln num]} ON [expr] GOTO [ln num] {, [ln num],...[ln num]} (5.8.14)	The value of the expression in the ON statement is an index to a list of line numbers at which the program is to continue execution (conditional branch)	ON A GOSUB 20, 60 ON A GOTO 100, 200, 300

Command, Function, Statement	Description	Example
PH 0. PH 1. (5.8.24)	Same as PRINT except that the values are output in hexadecimal with or without suppression of leading zeros	PH 0. 1000
PH 0.# PH 1.# (5.8.24)	Same as PH0. and PH1. except that the values are output over the unidirectional interface	PH 0.# 500
POP [var] (5.8.26)	Assigns the last values that were pushed onto the argument stack to the variables listed in the POP statement	POP A, B, C
PRINT [list of expressions] (5.8.22)	Outputs values in the expression list over the bidirectional interface	PRINT ”CP 521 BASIC”
PRINT # [list of expressions] (5.8.23)	Outputs values in the expression list over the unidirectional interface	PRINT # ”CP 521 BASIC”
PROG (5.7.3)	Stores the program currently in RAM on the memory submodule	PROG
PROG 1 (5.7.4)	Same as PROG but also stores the baud rate info. On a restart, PROG1 starts the program in RAM if the mode selector is set to RUN.	PROG 1
PROG 2 (5.7.4)	Same as PROG1 but starts program no. 1 on the memory submodule	PROG 2
PUSH [expr] (5.8.25)	Pushes the arithmetic expressions that follow the word PUSH onto the argument stack	PUSH C, B, A
RAM (5.7.1)	Select the program in RAM as current program	RAM
RDCLOCK [strvar] RDDATE [strvar] (5.13.2, 5.13.4)	Reads the time or date into ”strvar” and returns a read error, if any, as function value	X=RDCLOCK $(0)
READ [var] {, [var], ...} (5.8.6)	Reads data from a DATA statement line and assigns it to a variable	READ A
REM (5.8.27)	Identifies commentary lines	REM PRINT THE ANSWER
REORG (5.7.6)	Erases programs and closes the gaps between the remaining programs	REORG
RESTORE (5.8.6)	Reads data from a DATA statement line and assigns it to a variable	RESTORE

Command, Function, Statement	Description	Example
RETI (5.8.28)	Exits interrupt routines	RETI
RETURN (5.8.12)	Exits a subroutine and returns to the main program	RETURN
ROM [integer] (5.7.1)	Selects the program with the specified number from the memory submodule	ROM 15
RROM [integer] (5.8.30)	Starts the memory submodule-resident program with the specified number	RROM 5
RSTAT (5.9.4)	Returns a numeric value representing the status of the Receive buffer	X=RSTAT IF X>0 THEN PRINT ”Characters available”
RUN (5.6.1)	Sets all variables to 0 and starts the program at the first line number	RUN
SEND [expr] (5.9.7)	Enables fetching of the Send data	SEND 210
SETCLOCK [strvar] SETDATE [strvar] (5.13.1, 5.13.3)	”strvar” contains the time or date. An error status is returned as function value.	X=SETCLOCK $(0)
SGET [var1], ... ,[var4] (5.9.1)	Reads 4 words from the CP 521 BASIC's I/O area and assigns them to the specified variables. The values were forwarded to the CP from the CPU.	SGET W0, W1, W2, W3
SLEN strvar (5.12.5)	Returns the length of string ”strvar”	LNG=SLEN $(1)
SPUT [expr1], ... ,[expr4] (5.9.2)	Assigns the value of the expressions to the CP 521 BASIC's I/O area. These values can be read out over the CPU.	SPUT W0, W1, W2, W3
SSTAT (5.9.3)	Returns a numeric value representing the status of the Receive buffer	X=SSTAT
STOP (5.8.29)	Stops the program	STOP
STRING [expr] , [expr] (5.12.1)	Reserves memory space for strings	STRING 50, 10

Command, Function, Statement	Description	Example
T_ADJ [expr] (5.13.5)	Corrects the system clock at regular intervals	T_ADJ 100
TIME (5.8.21)	Assigns a value for the timer or queries the current time value	TIME=0 PRINT TIME
XBY ([expr]) (5.14.1)	Reads out the contents of specific addresses	A=XBY (0F000H)
XFER (5.7.2)	Transfers the current program to RAM for processing	XFER
”+” (5.12.2)	Concatenates two strings	$(1)=$(0)+”END”

List of available functions and operators

Function/ Operator	Description	Example
+	Addition	X=A+B
-	Substraction	PRINT A - B
*	Multiplication	A=B * X
/	Division	X=5/A
**	Exponentiation	X=2**4
.AND.	Logical AND	X=26.AND.2
.OR.	Logical OR	X=25.OR.0AH
.XOR.	Exclusive OR	X=A.XOR.C
NOT	Logical negation	X=NOT(X)
ABS	Absolute operand value	X=ABS(B)
INT	Integer portion of a no. or integer no.	X=INT(X)
SGN	Sign (of the number)	X=SGN(X)
SQR	Square root function	X=SQR(C)
RND	Pseudo random number betw. 0 and 1	X=RND
LOG	Natural logarithm	X=LOG(PI)
EXP	Exponential function	X=EXP(X)
PI	Constant =3,14 ...	X=PI**PI
SIN	Trigonometric functions — Sine	X=SIN (PI/4)
COS	Trigonometric functions — Cosine	X=COS (PI)
TAN	Trigonometric functions — Tangent	X=TAN (A)
ATN	Trigonometric functions — Arc tangent	X= ATN (C)

Appendix B:

B.1 Active and Passive Faults in Automation Equipment. . . B - 1

B.1 Active and Passive Faults in Automation Equipment

- Depending on the particular task for which the electronic automation equipment is used, both **active** as well as **passive** faults can result in a **dangerous** situation. For example, in drive control, an active fault is generally dangerous because it can result in unauthorized startup of the drive. On the other hand, a passive fault in a signalling function can result in a dangerous operating state not being reported to the operator.

- This differentiation of the possible faults and their classification into dangerous and non-dangerous faults, depending on the particular task, is important for all safety considerations in respect to the product supplied.

Warning

In all cases where a fault in automation equipment can result in severe personal injury or substantial damage to property, i.e. where a dangerous fault can occur, additional external measures must be taken or equipment provided to ensure or force safe operating conditions even in the event of a fault (e.g. by means of independent limit monitors, mechanical interlocks etc.).

Maintenance and Repair Procedure

If any measurement or testing work has to be carried out **on the CP 521 BASIC**, the rules and regulations set forth in the VBG 4.0 Accident Prevention Regulations" of the German Employees Liability Assurance Association must be observed, in particular §8 "Permissible exceptions when working on live parts".

Do not open up the CP 521 BASIC under any circumstances.

Repairs to an item of automation equipment may only be carried out by **Siemens service personnel** or **repair shops authorized by Siemens** to carry out such repairs

Index

Index

A

Acknowledgement for the last subframe 6-16
Address area 4-6
Addressing 4-6
Alternate branch 5-37
Argument stack 5-25, 5-78, 5-80
Array (field)
- dimension 5-47
Auto Baud routine 5-2, 5-37, 6-2

B

Backup battery 2-7, 2-8
BASIC interpreter 1-1, 5-1, 5-20
- error message 5-130, 7-2, 7-3
BASIC program 1-1
- interrupt 5-44
- load 5-15
- program structure 8-3
- save 5-13
- start 6-26
Battery
- backup 2-7, 2-8
- change 2-7
- failure 4-4
- test 4-4
Baud rate 5-2, 5-36, 5-40, 5-42
Bidirectional interface
- data output 5-71
- forwarding data 5-94, 5-95
- initialize 5-40
- receiving data 5-94, 6-39
Bidirectional TTY interface 2-6
- sending data 6-38
Bidirectional V.24 interface 2-6
- control signal 6-35
- receiving data 6-39
- sending data 6-38
Block 8-7
Buffer command 5-87
BUSY signal 5-43, 6-40

C

Cable routing 3-3
Carriage return 5-73
Clock
- adjustment 5-127
- date 5-126
- default 4-4
- default set 6-30
- error code 4-4
- integral 2-6, 4-1
- read out the current data 6-32
- returning current time 5-124
- set 6-33
- set date 5-125
- status 6-29
- test 4-4
COM Software 5-7
- CONTENTS menu 5-19
- DEFAULTS menu 5-10
- LOAD menu 5-16
- SAVE menu 5-14
- start 5-7
- start form 5-9
- TERMINAL EMULATION menu 5-12
- transfer 5-7

Command 5-26, 5-31
- auxiliary 5-86
- mode 5-6, 5-6

Communications interface 6-1
Confirmation message 6-25
Connection diagram 3-12
- TTY interface 3-15, 3-16, 3-17
- V.24 interface 3-12, 3-13, 3-14

Constant 5-21
Control signal
- bidirectional V.24 interface 6-35

Control stack 5-25
Coordination
- data 6-10
- receiving data 6-4
- request 6-9, 6-17, 6-20

CP 521 BASIC
- program 8-3

CP acknowledgement 6-3, 6-4, 6-10
- 1st subframe 6-12
- 2nd subframe 6-14
- coordination info 6-10
- request 6-21
- sending the last (43rd) subframe 6-16

CPU acknowledgement 6-5
- arrival of the 1st subframe 6-22
- 25th (last) subframe received 6-24

CPU request 7-1
- coordinate data transfer 6-9, 6-20
- sending data 6-11
- sending a frame 6-3
- sending the 1st subframe 6-11

CPU request
- sending the 2nd subframe 6-13
- sending the 43rd subframe (max.) 6-15
- set clock 6-33
- start program 6-27

D

Data
- enable 5-93
- forward to the PII 5-90
- input 5-59
- length 5-91
- read from the PIQ 5-89
- receive 6-38, 6-39
- send 6-38, 6-39, 6-40
- type 5-91

Data format 5-21, 5-41, 5-43, 5-73
- constant 5-21
- floating-point representation 5-21
- integer representation 5-21

Data frame
- forwarding to CP 6-6

Data interchange 4-2, 6-1
- direct access 6-28

Data output
- bidirectional interface 5-71
- unidirectional interface 5-76, 5-95

Data transfer 4-1
- bidirectional interface 6-34
- coordination 6-6, 6-9, 6-10, 6-17, 6-20
- CP 521 BASIC and I/O device 6-34

Data transfer
- CPU-CP 521 BASIC 6-3
- error flag 6-17
- receive 6-20
- receiving the 1st data block 6-21
- receiving the25th (last) subframe 6-23
- sending frames 6-6
- sending the 1st subframe 6-11
- sending the 2nd subframe 6-13
- sending the 43rd subframe (max.) 6-15
- sending with end character 6-16

Date 6-31
Default
- interface 5-40, 5-41
- set 6-30

D Sub socket connector 3-11
- pin assignment

Dyadic Operator 5-102

E

End character 6-9, 6-16, 6-17
Error code 7-5
- clock 4-4

Error flag
- BASIC interpreter 5-36
- data transfer 6-17

Error message
- BASIC interpreter 5-130, 7-2, 7-3
- BASIC mode 7-7

Expression 5-24
- relational 5-24

F

FB 1 8-13
FB 2 8-15
FB 3 8-17
FB ”IMPULS” 8-11
FB ”VERTEILE” 8-12
FB 200 ”Send” 8-19
FB 201 ”Receive” 8-27
Floating-point representation 5-21
Format statement 5-21
Forwarding data
- bidirectional interface 5-94, 5-95
- unidirectional interface 5-94

Frame 6-3, 6-4
- length 6-9
- receive 6-4, 6-18, 6-20
- receiving from the CP 6-18

Frame
- send 6-3, 6-5, 6-9
- sending with end character 6-16
- size 6-17

Function block
- FB ”IMPULS” 8-11
- FB 201 ”Receive” 8-27
- FB ”VERTEILE” 8-12

H

HANDLE
- Receive buffer 4-1, 5-88, 5-94, 5-99 - 5-101, 6-38, 6-39
- Send buffer 4-1, 5-87, 5-94, 5-95, 5-97, 6-38 - 6-40

Handshake 5-41, 5-43
- OFF mode 6-36
- ON mode 6-36

I

I/O device 6-34, 6-38, 6-40
- permissible 4-3
- receiving data 6-39

Initialization
- bidirectional interface 5-40, 5-42
- bidirectional serial interface 5-40
- interface 5-40
- serial interface 5-94, 6-2
- unidirectional interface 5-43
- unidirectional V.24 interface 5-42

Installation 3-1
Integer representation 5-21
Interface 4-2
- communications 6-1
- initialization 5-36, 5-40, 5-42, 5-43, 5-94, 6-2
- parameter 5-2, 5-3, 5-40, 5-41
- serial 1-1, 2-6, 6-1
- TTY 1-1, 2-6, 3-11, 6-38
- V.24 1-1, 2-6, 3-11, 5-2, 6-38

Interrupt 5-66, 5-83
- disable 5-39
- routine 5-66, 5-83

L

LEDs 2-8
Logarithm function 5-107
Logical Operator 5-104
Loop 5-48, 5-49, 5-51

M

Memory command 5-31
Memory submodule 2-5, 2-8, 5-36
- displaying the contents 5-18
- programming 5-17

Menu
- CONTENTS 5-19
- DEFAULTS 5-10
- LOAD 5-16
- SAVE 5-14
- TERMINALEMULATION 5-12

Mode switch 4-5

O

Operator 5-22
- dyadic 5-102
- logical 5-104
- priority 5-110
- unary 5-105, 5-107, 5-108

P

Parity 5-40, 5-43
PII process input image
Pin assignment 3-11
- D sub socket connector 3-11

PLC
- programming 8-7

POI process output image
Process input image 4-6
- forward data 5-90

Process interrupt 6-3, 8-12
Process output image 4-6
- read data 5-89

”PROG” command 5-3, 5-34 - 5-36
Program 8-1
- block 8-11
- continue 5-56
- delete 5-30
- erase 5-37
- line 5-20
- output 5-28
- resume 5-27, 5-55
- transfer 5-33, 5-34
Program loop
- exit 5-39
Program run
- abortion 5-26
- resume 5-27
- start 5-85
- stop 5-84
- terminate 5-50
Program structure
- BASIC program 8-3
- block 8-9
Programming 5-6
- CP 521 BASIC 8-3
- memory submodule 5-17
- PLC 8-7
Putting into operation 6-2

R

Real-time clock
- integral 2-7, 4-1
Receive 6-39
- bidirectional interface
- mailbox 4-1, 6-3, 6-4
Receive buffer 5-87, 5-92
- HANDLE 4-1, 5-84, 5-94,
Receive buffer
- HANDLE 5-99 - 5-101, 6-38, 6-39
Receive frame 6-4, 6-18, 6-20
- FB 3 8-17
Receive data 6-1, 6-38, 6-39
- bidirectional interface 5-94
Relational expression 5-24, 5-111
Request to Send 6-3
RESET 5-2, 5-37
Restart characteristics 4-3, 5-2, 5-3
RUN-Mode 5-6

S

Screen form
- start form 5-9
Send
- bidirectional interface 6-38
- mailbox 4-1, 6-3, 6-4
- unidirectional interface 6-40
Send buffer 5-87, 5-93
- HANDLE 4-1, 5-87, 5-94, 5-95, 5-97, 6-38, 6-39
Send data 6-1, 6-38, 6-39, 6-40
- CPU request 6-11
Send FB
- FB 200 ”Send” 8-19
Send frame 6-3, 6-5, 6-9
- FB 1 8-13
- FB 2 8-15
Serial interface 1-1, 2-6, 6-1
- initialization 5-94, 6-2

Slot 3-1, 4-6
Space
- bar 5-37
- key 5-2
Stack 5-25
- argument 5-25
- control 5-25
- reset 5-39
Statement 5-20, 5-38
Status
- byte 8-10
- last Receive request 5-91
- last Send request 5-90, 5-96
String 5-112
- combine 5-115
- length 5-118
- manipulate 5-118
- operation 5-112
- substring 5-116, 5-117
- variable 5-125, 5-113
Subframe 6-3, 6-4, 6-21
- number 6-3, 6-4, 6-12, 6-21
Subroutine 5-53, 5-78, 5-81
System clock
- adjustment 5-127
System control parameter 5-24, 5-129

T

Technical specifications 2-1, 2-4
Terminal emulation 5-11
Termination message 6-5
Test
- battery 4-4
Time 6-31
- set the clock 5-123
Timer
- enable 5-68
- read out 5-70
- stop 5-69
Trigonometric function 5-108
TTY interface 1-1, 3-11, 6-38
- bidirectional 2-6
- connection diagram 3-15 - 3-16, 3-17

U

Unary Operator 5-105, 5-107, 5-108
Unidirectional interface
- data output 5-76, 5-95
- forwarding data 5-94
- sending data 6-40
- V.24 2-6

V

V.24 interface 1-1, 2-6, 3-11, 5-2, 6-38
- bidirectional 2-6
- connection diagram 3-12 - 3-13, 3-14
- unidirectional 1-1, 2-6
Variable 5-22, 5-59, 5-60, 5-80
- allocate the value 5-62
- set to zero 5-38
- string 5-112, 5-113

An
Siemens AG
AUT E 148
Postfach 1963

D-92209 Amberg
Federal Republic of Germany

From:

Your Name: _____

Your Title: _____

Company Name: _____

Street: _____

City, Zip Code: _____

Country: _____

Phone: _____

Please check any industry that applies to you:

- ☐ Automotive
- ☐ Chemical
- ☐ Electrical Machinery
- ☐ Food
- ☐ Instrument and Control
- ☐ Nonelectrical Machinery
- ☐ Petrochemical
- ☐ Pharmaceutical
- ☐ Plastic
- ☐ Pulp and Paper
- ☐ Textiles
- ☐ Transportation
- ☐ Other _____

Remarks Form

Your comments and recommendations will help us to improve the quality and usefulness of our publications. Please take the first available opportunity to fill out this questionnaire and return it to Siemens.

Title of Your Manual: ----------

Order No. of Your Manual: ----------

Edition: ----------

Please give each of the following questions your own personal mark within the range from 1 (very good) to 5 (poor).

1. Do the contents meet your requirements? ☐
2. Is the information you need easy to find? ☐
3. Is the text easy to understand? ☐
4. Does the level of technical detail meet your requirements? ☐
5. Please rate the quality of the graphics/tables: ☐

Additional comments: